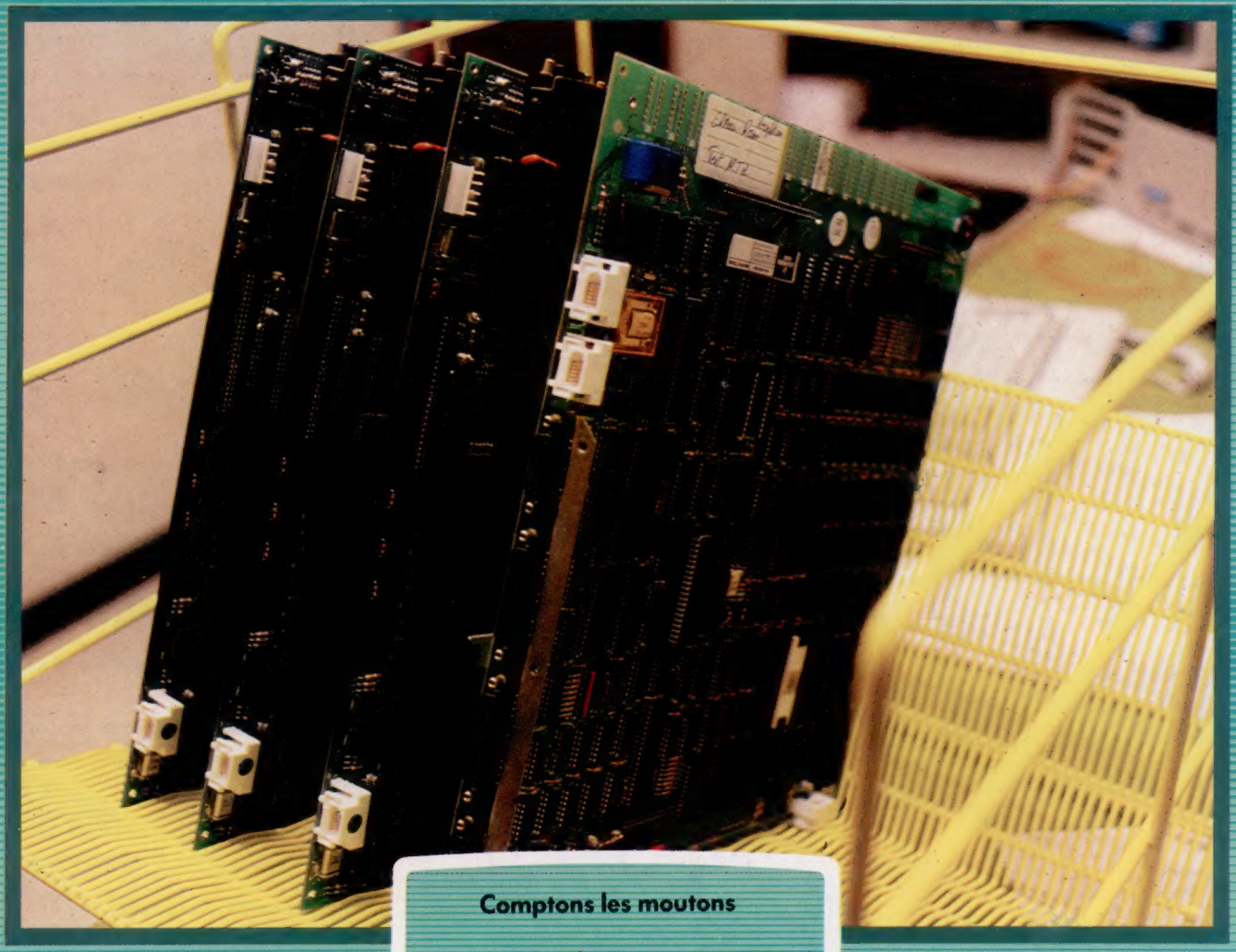
ABC INFORMATIQUE

N°60

COURS D'INFORMATIQUE PRATIQUE ET FAMILIALE



Comptons les moutons

Champ de manœuvre

Le Commodore Plus/4

Contrôler la puissance

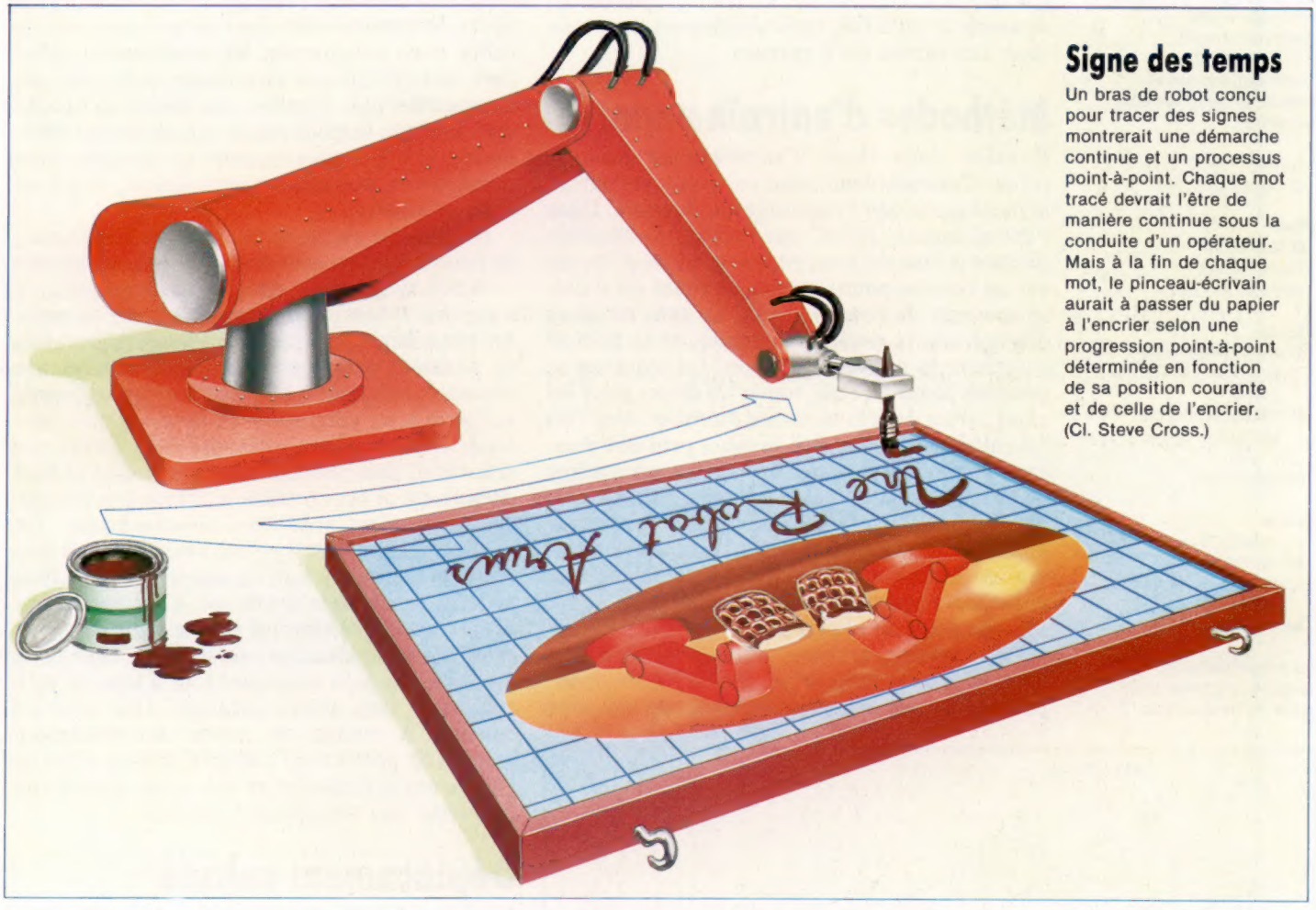
EDITIONS ATLAS

M 6062-60-12,50 F 90 FB - 3,80 FS - $ 1.95

Page manquante
(publicité et colophon)

Déplacements calculés

Nous avons étudié les méthodes de déplacement des robots, puis la conception de leurs « bras » et « mains ». Maintenant, abordons la manière de les programmer pour des tâches « intelligentes ».



Signe des temps

Un bras de robot conçu pour tracer des signes montrerait une démarche continue et un processus point-à-point. Chaque mot tracé devrait l'être de manière continue sous la conduite d'un opérateur. Mais à la fin de chaque mot, le pinceau-écrivain aurait à passer du papier à l'encrier selon une progression point-à-point déterminée en fonction de sa position courante et de celle de l'encrier. (Cl. Steve Cross.)

L'idée d'un bras intelligent peut paraître, à première vue, illogique. La forme d'intelligence étudiée ici n'a rien à voir avec celle, ultra-sophistiquée, de l'homme. Prenons l'exemple d'un acte humain simple. Vous êtes assis à une table vide, à l'exception d'un petit objet situé à gauche que vous devez déplacer vers la droite. Deux formes d'intelligence sont ici en cause : la première implique la perception de la table, de l'objet et la décision de déplacer ce dernier de la gauche vers la droite. Cela suppose un certain degré de conscience et met en jeu des concepts tels que « intention » et « objectif ». L'intelligence dont nous avons besoin ici est d'un degré bien inférieur à celle qui est nécessaire pour déplacer votre bras et votre main une fois la décision prise d'effectuer la tâche : déplacer votre main vers la droite et vous assurer qu'elle tient bien l'objet avant de le poser au bon moment.

Entraînement humain

Cela peut vous paraître à la fois facile et évident. Si vous doutez de l'intelligence dans cet acte, observez bien la manière de procéder d'un petit enfant : souvent il ne parviendra pas à saisir l'objet du premier coup, il le placera ailleurs que là où il doit aller, et, plus généralement, manifestera une hésitation sur ce qui lui est demandé. L'enfant s'exerce à acquérir l'intelligence active nécessaire pour déplacer les bras et les mains dans un univers à trois dimensions qui lui est encore étranger. Lorsqu'il aura appris à maîtriser ses mouvements, ils lui paraîtront presque automatiques. Il cessera alors d'y penser et d'y associer un effort d'intelligence. Le bras du robot en reste au stade du petit enfant : il est équipé pour la tâche requise, mais il doit apprendre comment la réaliser automatiquement.

Géométrie à deux articulations
En se déplaçant point-à-point, le bras à deux articulations doit effectuer une rotation sur son pivot (angle R), et modifier les angles des articulations de son épaule (E) et de son coude (C). Si les coordonnées cartésiennes des points courants et de destination sont (X1,Y1,Z1) et (X2,Y2,Z2), les modifications seront calculées de la manière suivante :

A1 = SQR(X1 ^ 2 + Y2 ^ 2 + Z1 ^ 2)
A2 = SQR(X2 ^ 2 + Y2 ^ 2 + Z2 ^ 2)

Pivot
R1 = ARCTAN(Y1/X1)
R2 = ARCTAN(Y2/X2)
Modification = (R2 – R1)

Épaule
S1 = ARCCOS(Z1/A1) + ARCCOS ((A1 ^ 1 + B ^ 2 – AB ^ 2)/ (2*A1*B))
S2 = ARCCOS(Z2/A2) + ARCCOS ((A2 ^ 2 + B ^ 2 – AB ^ 2)/ (2*A2*B))
Modification = (2 – S1)

Coude
E1 = ARCCOS((B ^ 2 + AB ^ 2 – A1 ^ 2)/(2*B*AB))
E2 = ARCCOS((B ^ 2 + AB ^ 2 – A2 ^ 2)/ (2*B*AB))
Modification = (E1 – E2)

où B représente la longueur du bras, et AB, celle de l'avant-bras.

La méthode la plus simple consiste à apprendre au bras à réaliser des tâches spécifiques en le guidant tout au long d'une séquence de déplacements, et en le forçant pratiquement à s'en souvenir. Cette méthode est utilisée avec de nombreux robots industriels. L'opérateur prend littéralement le robot par la main pour le guider à travers l'ensemble des étapes. Pour l'opérateur, l'avantage est manifeste, car il n'a pas besoin de savoir comment le bras articulé fonctionne. Il doit seulement savoir quelles séquences faire suivre au robot. De son côté, le robot n'a pas besoin de savoir ce qu'il fait, mais simplement de se souvenir des tâches qu'il exécute.

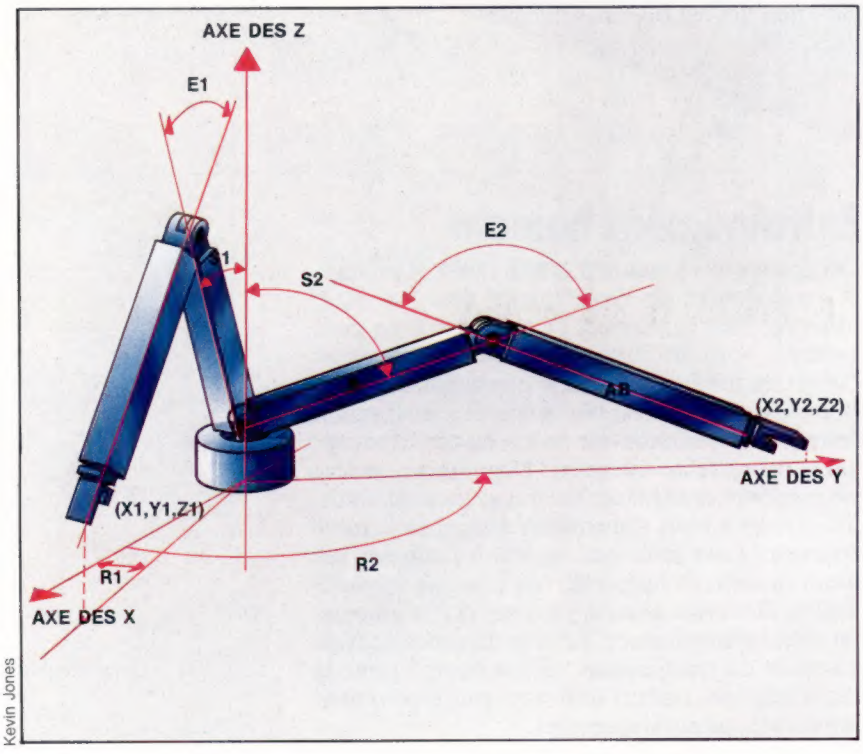
Méthodes d'entraînement

Il existe deux types d'entraînement pour le robot : l'entraînement point par point, et l'entraînement global sur l'ensemble du parcours. Dans l'entraînement point par point, l'opérateur déplace le bras sur une certaine position et appuie sur un bouton pour signifier au robot qu'il doit se souvenir de cette position. Le bras est alors déplacé sur la position suivante, et le bouton enregistre la nouvelle position. La séquence se poursuit jusqu'à ce que toutes les étapes aient été enregistrées dans la mémoire du robot. Une fois cet entraînement terminé, le robot peut être commuté en mode « playback », et il est alors à même de se déplacer d'un point à l'autre comme il l'a appris. Avec l'entraînement global, l'opérateur guide le robot à travers toute la séquence à suivre et le robot enregistre l'ensemble de la séquence. En mode retour *(playback)*, le robot reprend toute la séquence.

Pour revenir à notre exemple de l'objet à déplacer sur la table d'un côté à l'autre, nous pouvons utiliser la méthode d'entraînement point par point. Cette méthode est souvent utilisée avec les robots qui doivent effectuer des tâches du type « prendre et transporter » — déplacer un objet d'un endroit à l'autre. En revanche, un robot destiné à peindre au pistolet suivra avantageusement la méthode globale qui évite les interruptions.

Il nous faut maintenant envisager la manière dont le robot retient les séquences à suivre. Il enregistre, en fait, à l'aide de ses capteurs internes, la position de toutes ses articulations pendant la phase d'entraînement. Cela est souvent réalisé en relevant le résultat en sortie des encodeurs de transmission dans les articulations du robot et en enregistrant les mouvements effectués, soit directement en mémoire soit, pour des sauvegardes plus durables, sur bande ou sur disque. Lorsque le mode retour *(playback)* est sélectionné, le robot peut rappeler les données et les convertir en mouvements articulaires, ce qui est assez complexe.

De manière surprenante, le robot a beaucoup de facilité à se souvenir d'un mouvement continu — il doit simplement suivre la route exacte qu'il a apprise. Il faut cependant sauvegarder un nombre considérable de données. Plusieurs centaines de positions sont souvent nécessaires pour une séquence, alors que quelques positions seulement suffisent pour mémoriser un mouvement « pas-à-pas ». La deuxième difficulté vient du fait que si le robot doit pouvoir effectuer le mouvement doucement et avec précision, il faut que ses articulations soient sollicitées simultanément. Un robot de peinture doit pouvoir déplacer son bras selon les trois axes, tout en manœuvrant les trois articulations terminales de son « poignet » pour diriger avec précision les jets de peinture. Cela signifie que l'ordinateur, qui dirige le robot, doit agir très vite pour actionner tour à tour les articulations, sans aucun décalage. Une solution consiste à mettre en œuvre simultanément jusqu'à six processeurs distincts, chacun ayant en charge une articulation et une seule, dans le but d'obtenir une simultanéité parfaite.



Kevin Jones

Déplacement calculé

Un robot pas-à-pas a la tâche plus rude. Bien qu'il sache où il doit aller, il ne sait pas comment y aller. Il serait possible de manœuvrer chaque articulation jusqu'à obtenir la position voulue, mais ce serait une perte de temps et d'énergie. Il serait de loin préférable pour le robot de calculer sa route d'un point à l'autre : il pourrait alors effectuer son déplacement en un seul mouvement, à la façon d'une personne humaine. Mais les calculs sont aussi trop complexes, car les coordonnées cartésiennes sont nécessaires pour déplacer la main en ligne droite d'un point à un autre. Les positions du bras sont en outre définies selon un système de coordonnées différentes, par exemple des coordonnées circulaires. C'est pourquoi le robot doit être à même de résoudre des problèmes géométriques complexes pour fonctionner de manière satisfaisante. Dans le cas de robots industriels déplaçant sur de grandes distances des objets pesant plusieurs centaines de

tonnes, le choix de la meilleure trajectoire permet d'économiser de manière considérable du temps et de l'énergie.

Un autre problème concernant le déplacement pas-à-pas est la dynamique même du mouvement. Si vous étendez votre bras pour saisir quelque chose, vous observerez qu'il se déplace lentement au départ, pour atteindre la vitesse maximale en milieu de course et ralentir doucement avant d'atteindre son but. Pour une telle tâche, les avantages d'un bras de robot sont considérables. Il peut parfaitement démarrer son mouvement à toute vitesse, maintenir cette vitesse maximale et s'arrêter net en fin de séquence. Une grande contrainte s'exerce sur le bras et suppose davantage de puissance que pour un bras qui accélère et freine doucement. Cela signifie également que le bras court un risque trop grand s'il doit, en fin de course, prendre un objet délicat. Un simple déplacement malencontreux de l'objet à saisir est alors catastrophique. Le robot doit donc concilier vitesse optimale et trajectoire parfaite.

Chorégraphie robotique

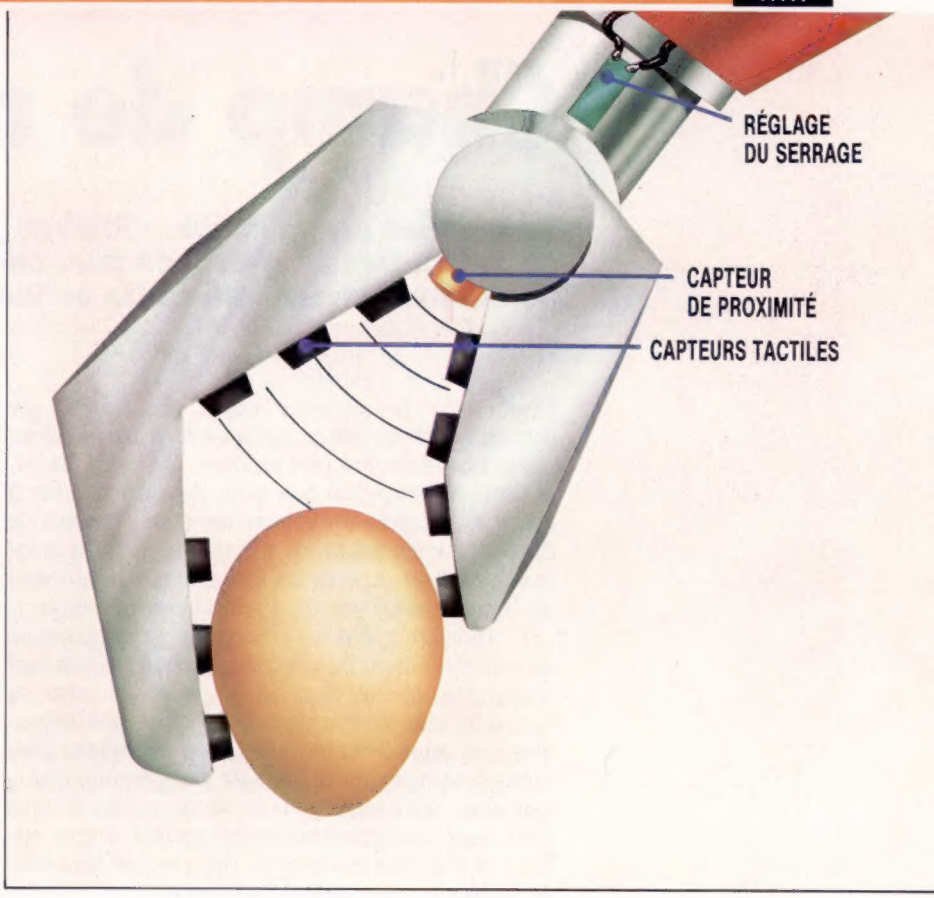
Même lorsque le bras a été programmé pour suivre un ensemble précis de mouvements, il peut apparaître, lors de la répétition de la séquence, que les mouvements enregistrés ne correspondent pas tout à fait aux tâches demandées. Cela peut être dû à une erreur humaine, à un léger changement dans la nature du travail à effectuer, ou à un travail du robot fait en conjonction avec d'autres bras articulés (ses bras entrent alors en collision avec d'autres bras). La démarche qui permet d'éviter une telle situation s'appelle « chorégraphie robotique ». Il faut donc mettre au point une méthode pour éditer la séquence. Cela peut se faire en sauvegardant les mouvements sous la forme d'une liste de mouvements enchaînés, dans laquelle chaque position est sauvegardée et suivie de l'adresse où l'on peut trouver la position suivante. Pour la séance initiale d'entraînement, cette adresse est celle de la position suivante dans la liste. Si la séquence doit être éditée, le bras peut être déplacé sur la position où des corrections doivent être faites, puis être arrêté de manière à ce qu'une nouvelle séquence soit insérée.

Une autre méthode pour rendre les bras articulés plus intelligents est d'utiliser un ensemble d'instructions programmées stockées dans l'ordinateur. Pratiquement, chaque robot possède sa propre méthode de programmation et met en œuvre un langage de programmation différent pour contrôler ses mouvements. Néanmoins, un langage est nécessaire pour que le programmeur puisse utiliser des commandes du genre LOGO afin de spécifier des déplacements dans trois dimensions et donner des instructions au « poignet » et à la « main », telles que « prendre ou « lâcher ».

Le problème est semblable à celui de l'entraînement du robot en mode pas-à-pas, et de nombreux facteurs sont à prendre en compte. Par exemple, si le bras doit avancer de 10 unités, la meilleure méthode est de modifier l'articulation de l'épaule, de sorte que le bras pourra aller plus loin. L'épaule doit alors se déplacer vers le haut selon un arc de cercle, et un déplacement compensateur vers le bas doit intervenir au niveau du coude. Nous voyons donc que cette instruction destinée à un simple mouvement doit se décomposer en deux séquences distinctes d'instructions destinées à deux articulations différentes.

D'autres problèmes se posent lorsque le bras doit prendre des objets. Aussi bien positionné que le bras puisse être, on peut difficilement être certain qu'il est bien à même de saisir l'objet, surtout lorsque ce dernier est de forme asymétrique. Il faut donc pourvoir le bras d'une main intelligente. Celle-là devra sentir la présence ou l'absence de l'objet, la distance à laquelle il se trouve et la force qu'elle devra exercer lorsqu'elle essaiera de le prendre. Ces problèmes seront résolus en équipant la main de capteurs de proximité, de capteurs tactiles et de force. Ces capteurs renvoient des informations sur l'action de la main (*feedback*) et permettent à l'ordinateur de contrôle d'effectuer les corrections nécessaires.

En prenant en compte tous ces problèmes, il est possible de concevoir un robot doté d'un certain degré d'intelligence. Néanmoins, aucun robot n'a encore pu être créé pour jouer au tennis, par exemple. La raison en est que l'intelligence du bras ne suffit pas. Le robot devrait, pour pouvoir rattraper une balle, en apprécier la direction, la hauteur, la force, et tenir compte d'une foule d'autres critères de positionnement, sans compter la résolution extrêmement complexe de problèmes de trajectoires. Pour cela, il faut beaucoup plus qu'un bras intelligent.



Doucement...
Faire prendre un œuf par un robot est un très bon test de recherche pour les capteurs et pour contrôler la rétroaction du bras.
Le capteur de proximité de la pince doit vérifier que l'œuf est assez près; les mâchoires peuvent alors commencer à se resserrer jusqu'à ce que les capteurs du toucher indiquent le contact avec l'œuf. Le résultat en sortie des capteurs du toucher doit être comparé à celui du capteur de proximité, lorsque la pince se referme sur l'œuf et que le bras se lève. Une chute brutale de la proximité signifie que l'œuf est en train de s'échapper. Aussi les mâchoires doivent-elles se serrer jusqu'à atteindre une limite prédéterminée de serrage, ou jusqu'à ce qu'une chute soudaine de résistance indique que la coquille de l'œuf se déforme avant de casser. (Cl. Steve Cross.)

Champ de manœuvre

De simples programmes utilitaires peuvent être écrits entièrement en BASIC. Des programmes plus complexes comportant des détails nombreux doivent être écrits en langage machine.

Pour faire fonctionner notre programme de recherche de variables, nous l'avons fusionné au programme devant être analysé. De cette façon, la seule information que nous devions fournir à partir du système d'exploitation était l'adresse de départ du programme BASIC; la fin du programme était repérée en recherchant le numéro de ligne le plus bas du programme utilitaire.

L'utilitaire que nous créons est un programme de remplacement de variable. Ce programme est très pratique : si vous avez utilisé un nom de variable dans un programme et que vous découvrez que celui-ci est incorrect, vous imaginez aisément le temps que ce type de programme pourrait vous faire gagner. Il en va de même si vous avez écrit un programme susceptible d'être utilisé par d'autres personnes, dans lequel les noms de variables sont peu évocateurs.

Dans cette partie, nous devons mettre le programme utilitaire dans une section séparée de mémoire, isolé du programme sur lequel il travaille. Nous devons trouver une méthode différente pour déterminer la fin du programme BASIC et une façon d'accueillir deux programmes BASIC en même temps dans l'ordinateur.

Les trois ordinateurs que nous examinons — le BBC Micro, le Commodore 64 et le Sinclair Spectrum — utilisent un jeu de pointeurs pour indiquer au système d'exploitation et à l'interpréteur BASIC où trouver les programmes BASIC et les variables, etc.

Sur le BBC Micro, il y a quatre pointeurs importants : PAGE et TOP, qui mémorisent les adresses de début et de fin du programme BASIC; LOMEM qui mémorise l'adresse de départ des variables BASIC; HIMEM, qui mémorise l'adresse finale de la zone BASIC. Ces quatre pointeurs sont stockés comme variables BASIC intégrées, et nous pouvons lire ou modifier leurs valeurs au moyen de simples instructions BASIC. Si nous avons un simple programme BASIC en mémoire et que nous désirons en ajouter un autre, nous donnons une valeur plus élevée à PAGE qu'à TOP — à l'aide de la commande OLD pour remettre à zéro TOP et LOMEM. Nous pouvons alors ajouter le nouveau programme sans modifier l'original. Nous passons d'un programme à l'autre en donnant de nouvelles valeurs à PAGE et à HIMEM en utilisant la commande OLD.

Dès que le programme utilitaire fonctionne, les valeurs des pointeurs correspondent au programme utilitaire; pour permettre à celui-ci de trouver le début et la fin du programme à modifier, nous devons copier les valeurs initiales dans une zone de mémoire qui ne sera pas modifiée lorsque nous changerons les programmes. Il est aussi possible de trouver la fin d'un programme en utilisant un marqueur placé par l'interpréteur BASIC. Il s'agit simplement d'un octet contenant une valeur de 128 ou plus, suivant immédiatement le caractère retour de chariot placé à la fin de la dernière ligne du programme. Cet octet et le suivant seront interprétés comme les octets HI et LO du prochain numéro de ligne. Puisque l'octet HI de ce numéro est 128 ou plus, cela donne un numéro de ligne de 32768 (256 × 128) ou plus. Comme le numéro de ligne valide le plus élevé est 32767, nous pouvons être certains d'avoir trouvé le marqueur de fin de programme et non seulement un autre numéro de ligne.

Le Commodore 64 utilise plusieurs pointeurs, stockés dans la mémoire de page zéro, pour indiquer diverses parties de la zone de programme BASIC. TXTTAB, aux adresses 43 et 44, pointe le début du programme BASIC; VARTAB, ARYTAB, STREND, FRETOP et FRESPC, aux adresses 45 à 54, pointent diverses sections de la table de variables; et MEMSIZ, aux adresses 55 et 56, pointe la fin de la zone BASIC. Il est possible de changer ces pointeurs afin de créer une zone distincte où exécuter un programme BASIC au moyen de la commande POKE. Cependant, il est préférable d'utiliser un court programme en langage machine : puisqu'il est plus direct, il réduit les risques de blocage de l'ordinateur à la suite d'une erreur de frappe.

Dans le Commodore 64, la fin d'un programme BASIC est indiquée par 2 octets renfermant des zéros placés immédiatement après l'octet zéro qui désigne la fin de la dernière ligne du programme. Pour trouver la fin d'un programme, il suffit d'examiner les chaînes de pointeurs placées au début de chaque ligne jusqu'à la rencontre d'un pointeur zéro.

La création de ce programme est plus compliquée sur le Spectrum. Au lieu d'une zone de mémoire distincte pour le programme BASIC, un seul bloc de mémoire continu se présente. Il contient non seulement le programme et les variables BASIC, mais aussi les zones d'espace de travail de mémoire utilisées par le système d'exploitation et par l'interpréteur BASIC.

Avec cette disposition de mémoire, il est difficile, sinon impossible, de placer deux programmes BASIC dans la principale zone de travail; nous ferons donc une copie de notre programme au-dessus de RAMTOP et le modifierons à cet endroit. Mais nous devons également récupérer

le programme et le reloger dans la principale zone de programme après l'avoir modifié. Nous aurons besoin d'un programme en code machine pour le faire.

Le manuel du Spectrum nous apporte beaucoup d'informations sur le mode de stockage d'un programme BASIC comme sur l'utilisation des diverses zones de mémoire. Cependant, en raison du grand nombre de sections différentes de la zone de travail, et de leur mode de déplacement, il est difficile d'écrire des programmes utilitaires sans utiliser des sous-programmes en langage machine logés en ROM. Si vous désirez écrire des utilitaires évolués sur le Spectrum, il existe d'excellents ouvrages où est expliqué comment fonctionnent toutes les routines en ROM. Deux des sous-programmes les plus importants logés en ROM destinés à être utilisés dans les programmes utilitaires sont les routines qui ouvrent et sollicitent de l'espace dans la zone de travail.

Expérience avec basic

Vous pouvez modifier le contenu d'un programme pendant l'exécution, mais vous devriez d'abord sauvegarder le programme, puisqu'il arrive fréquemment que cela bloque le système. Utilisez le programme Moniteur qui vous permet d'examiner et de modifier le contenu de la mémoire. Insérez quelques lignes REM supplémentaires au début du programme et essayez ces suggestions :

- Trouvez le début de la zone de texte BASIC et inspectez le programme Moniteur en mémoire jusqu'à ce que vous puissiez identifier les lignes de programme.
- Changez les valeurs des octets placés après un REM élémentaire, puis quittez le programme et listez la ligne modifiée.
- Essayez de mettre une valeur supérieure à 127 dans une ligne REM ; de nouveau quittez et listez.
- Modifiez d'une ligne les octets de numéro de ligne : les conséquences sont imprévisibles.
- Vous pouvez modifier les octets de longueur de ligne — essayez de diminuer d'abord la longueur indiquée —, mais vous devez insérer un nouveau marqueur de fin de ligne à l'octet indiqué.
- Sur le Commodore 64, vous pouvez changer les octets d'adresse de chaîne : essayez de remplacer une adresse de chaîne d'une ligne par l'adresse de chaîne de la ligne suivante.
- Si vous vous sentez plus ambitieux, consultez votre manuel et explorez la zone de stockage des variables. Elle commence habituellement à l'endroit où se termine la zone de texte BASIC. Il y a jusqu'à six types de variables, chacun ayant son propre format de stockage : les variables numériques, les tableaux numériques, les variables de nombres entiers, les tableaux de nombres entiers, les variables de chaînes, les tableaux de chaînes. Les formats de variables de chaînes et de nombres entiers sont les plus simples, les données de tableaux numériques sont les plus compliquées.
- Vous pouvez changer les valeurs élémentaires dans des lignes de programme, ce qui changera le mot de commande. Si vous faites cette intervention sur une ligne en cours d'exécution avec le programme Moniteur, vous introduirez un paradoxe potentiellement important dans l'interpréteur.

Comment sont stockés les programmes basic

La plupart des micros obéissent au même format de stockage dans la zone de programme BASIC. Chaque ligne de programme commence par les données de ligne : numéro de ligne BASIC dans 2 octets et une certaine information concernant la longueur de la ligne. Le texte du programme est stocké sans modification importante, bien que les mots clés BASIC soient remplacés par des numéros de code à 1 octet nommés « jetons ».

500 REMEXEMPLE****

Version du BBC

13	HI 1	LO 244	17	32	REM	*	*	E	X	E	M	P	L	E	*	*
	Marqueur de début de ligne	Numéro de ligne	Longueur de ligne		Jeton	Texte du programme										

Version du Commodore 64

LO 18	HI 8	LO 244	HI 1	REM	*	*	E	X	E	M	P	L	E	*	*	0
	Adresse de chaîne	Numéro de ligne		Jeton	Texte du programme										Marqueur de fin de ligne	

Version du Spectrum

HI 1	LO 244	LO 13	HI 0	REM	*	*	E	X	E	M	P	L	E	*	*	13
	Numéro de ligne	Longueur de ligne		Jeton	Texte du programme										Marqueur de fin de ligne	

Dans ce format, 13 (code ASCII de [RETURN]) est un marqueur de début de ligne ; il est plus commodément placé à la fin d'une ligne. La longueur de ligne n'a que 1 octet, ce qui limite les lignes de programme à 255 caractères. L'espace apparaissant immédiatement après le numéro de ligne a été stocké.

Les octets d'adresse de chaîne renferment l'adresse à 2 octets du premier octet de la prochaine ligne de programme. La zone de texte de programme BASIC commence à l'adresse 2049, et cette ligne a une longueur de 17 octets, l'adresse de départ de la ligne suivante est donc 2066.

Pour le Spectrum, la longueur de ligne occupe 2 octets ; une simple ligne de programme peut donc avoir une longueur de 65 535 caractères ! La longueur de ligne est ici 13 — les octets faisant partie de la ligne incluant l'octet de fin de ligne, mais n'incluant pas les octets de données de ligne.

Liz Dixon

Compter les moutons

Les capacités de LOGO sur Atari sont particulièrement brillantes. Ses commandes définissent des figures, déterminent leur vitesse et permettent un grand choix de couleurs.

LOGO sur Atari peut mettre en œuvre quatre figures graphiques (ou lutins) de 0 à 3, 0 étant la tortue par défaut. Le manuel Atari les appelle « tortues » plutôt que figures, et nous ferons de même.

DIRE 1 fait de la tortue 1 la tortue courante. La tortue 1 obéira donc à toutes les commandes que vous passerez. Essayez par exemple :

```
DIRE 1
AV 40
DR 90
DIRE 2
AR 40
DR 90
DIRE 3
DR 90
```

Mais vous pouvez avoir plus d'une tortue courante :

```
DIRE [1 2 3]
AV 50
```

Les trois tortues désignées obéiront aux commandes.

Les quatre tortues ont au départ la forme de la tortue 0 (jusqu'à ce qu'elles soient définies explicitement).

EDFORME 1 fera éditer la forme 1 par l'éditeur. Les touches curseur vous permettront de vous déplacer librement sur l'écran. La barre d'espacement transformera un pavé graphique vide en pavé rempli d'une couleur, et un pavé plein en un pavé vide. La touche d'échappement <ESC> permet de définir une forme dessinée à l'écran. La commande ATTRIBUE 1 permet d'attribuer la forme 1 aux tortues 1, 2, 3 (les tortues courantes).

DEMANDE permet d'effectuer une commande par une tortue sans changer la tortue courante :

```
DEMANDE 1[AV 20]
```

Seule la tortue 1 a bougé. Si vous tapez AV 20, vous verrez que les trois tortues ont avancé, car elles sont toujours numérotées 1, 2 et 3.

Outre une direction et une position, les tortues peuvent recevoir une vitesse DONNEVITESSE 30 donne à la tortue courante la vitesse 30 selon sa direction initiale. Les tortues garderont la même vitesse de déplacement jusqu'à ce que cette dernière soit modifiée. Vous pouvez exécuter des procédures et créer des dessins, la vitesse ne change pas. Pour arrêter les tortues, donnez-leur une vitesse 0. Le fait de passer dans le mode éditeur les arrêtera également, puisque l'éditeur, utilisant la même zone mémoire, détruit l'affichage graphique.

Atari autorise cent vingt-huit nuances. Vous pouvez choisir la couleur du fond, celle du crayon et celle de la tortue. Par exemple, DONNECOULEURFOND 92 donnera au fond la couleur verte. DONNECOULEURCRAYON 0 23 donnera la couleur orange au crayon. 23 est le code pour la couleur orange, et 0 est le code pour le numéro du crayon. Avec le LOGO d'Atari, il est possible d'utiliser trois crayons, mais nous n'utiliserons ici que le crayon 0 (le crayon par défaut). Ici, DONNECOULEUR 7 donnera la couleur blanche à la tortue courante.

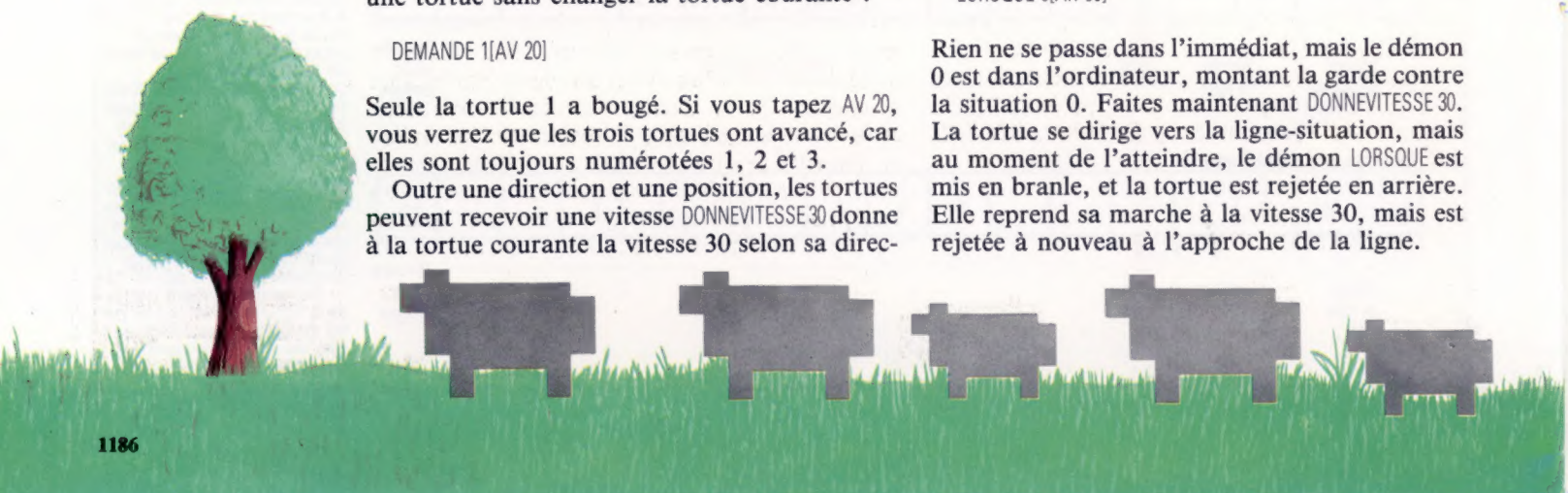
L'aspect le plus original de LOGO sur Atari est l'utilisation des « DÉMONS ». LOGO peut détecter jusqu'à vingt et un types de collisions et autres situations. Essayez par exemple ceci :

```
COLLISION
DIRE 0
POSEPLUME
AV 50
LÈVEPLUME
DR 90
AV 100
DR 90
AV 20
DR 90
```

Mettez alors en place un démon LORSQUE par la commande suivante :

```
LORSQUE 0[AR 50]
```

Rien ne se passe dans l'immédiat, mais le démon 0 est dans l'ordinateur, montant la garde contre la situation 0. Faites maintenant DONNEVITESSE 30. La tortue se dirige vers la ligne-situation, mais au moment de l'atteindre, le démon LORSQUE est mis en branle, et la tortue est rejetée en arrière. Elle reprend sa marche à la vitesse 30, mais est rejetée à nouveau à l'approche de la ligne.



Ce démon LORSQUE reste en activité jusqu'à ce que vous le supprimiez par LORSQUE 0 []. Taper COLLISION supprimera tous les démons. Mais un message d'erreur ou l'utilisation de l'éditeur les supprimera aussi bien.

Comme il est fastidieux d'avoir à se rappeler tous les codes de collisions, deux primitives vous y aideront : AU.DELÀ <numéro de la tortue> <numéro du crayon> vous communiquera le numéro de la collision entre la tortue et une ligne tracée par le crayon. TOUCHE <numéro de la tortue 1> <numéro de la tortue 2> donne le numéro du démon chargé de la collision entre ces tortues.

Mettre la tortue en cage

Voici une procédure pour mettre en cage une tortue. Chaque fois qu'elle heurte les parois (LORSQUE AU.DELÀ 0 0), un démon appelle la procédure TOURNE. Cela fait revenir la tortue en arrière de 10 unités, après quoi elle effectue un tournant au hasard (AU.HASARD, avec un numéro, N, produit un nombre aléatoire compris entre 0 et N-1 inclus).

```
POUR PIÈGE
   TRACE.PIÈGE
   RETOUR
   LORSQUE AU.DELÀ 0 0 [TOURNE]
   DONNEVITESSE 50
FIN

POUR TRACE.PIÈGE
   COLLISION
   LÈVEPLUME
   POSITIONNE [-50 -50]
   POSEPLUME
   CARRÉ
   LÈVEPLUME
FIN

POUR CARRÉ
   RÉPÈTE 4 [AV 100 DR 90]
FIN

POUR TOURNE
   AR 10
   DR AU.HASARD 45
FIN
```

Les démons peuvent également être affectés à la manette de jeu. Parmi les vingt et une situations que nous avons citées, la situation 3 se produit lorsque l'on appuie sur la manette, et la situation 15 lorsque sa position est modifiée. La commande MAN 1 produit un nombre compris entre −1 et 7, correspondant à la position de la manette (relié par le port 2). Définissez DIRECTIONMANETTE de la manière suivante :

```
POUR DIRECTIONMANETTE
   SI (MAN 1) < 0 [STOP]
   DEMANDE 0 [DONNEDIRECTION 45 *
      MAN 1]
   FIN
```

Mettez ensuite la tortue en marche par DONNEVITESSE 50, et, en dernier lieu, placez un démon LORSQUE :

```
LORSQUE 15 [DIRECTIONMANETTE]
```

La manette peut alors être utilisée pour diriger la tortue 0.

Vous pouvez donner plus d'une commande LORSQUE à la fois, mais elles ne seront pas actives simultanément. Lorsqu'un démon est « occupé » (c'est-à-dire lorsque se réalise une situation), les autres sont inactifs. Ce qui signifie que certaines collisions peuvent ne pas être détectées.

La solution est de faire mettre les vitesses à zéro par les démons, et de faire s'exécuter une procédure continue qui traite des vitesses. Voici comment adapter votre programme précédent à cette technique :

```
POUR PIÈGE
   TRACÉ.PIÈGE
   RETOUR
   LORSQUE AU.DELÀ 0 0 [DONNEVITESSE 0]
   SURVEILLANCE
FIN

POUR SURVEILLANCE
   SI :VITESSE = 0 [DÉTECTION]
   SURVEILLANCE
FIN
```

Dans cette procédure, VITESSE donne la vitesse de la tortue courante. La procédure DÉTECTION doit déterminer la situation qui vient de se produire, prendre les mesures nécessaires et restaurer les vitesses appropriées. Dans le cas présent, il n'y a qu'une situation qui nous préoccupe, mais le principe vaut pour la méthode de programmation.

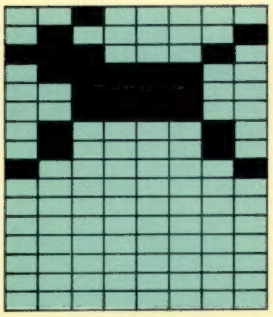
```
POUR DÉTECTION
   SI CONDITION AU.DELÀ 0
   0 ALORS [TOURNE]
   DONNEVITESSE 50
FIN
```

La commande CONDITION suivie d'un nombre donne un résultat VRAI si la situation correspond au nombre indiqué. CONDITION ne peut détecter une situation que lorsque LOGO exécute la ligne où cette dernière se trouve.

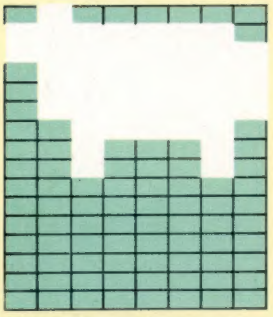
Encercler les moutons

Voici un jeu qui met en œuvre beaucoup des caractéristiques que nous avons décrites. Le joueur utilise la manette de jeu pour contrôler le déplacement d'un chien qui court après deux moutons dans un champ. Lorsque les moutons se heurtent au grillage, ils reviennent en arrière et changent de direction. Lorsque les moutons se heurtent entre eux, ils tournent dans une direction prise au hasard. Lorsque le chien attrape un mouton, ce dernier tourne de 90° à droite. En

Kevin Jones



Canis familiaris — le chien



Liz Dixon

Ovis aries — le mouton

Début du jeu

Le jeu en pleine action

Ian McKinnel

Retour au bercail

appuyant sur le bouton de la manette de jeu, on dessine une petite cage dans le coin inférieur gauche du champ. En appuyant à nouveau sur ce bouton, on efface la cage. Le travail du chien consiste à traquer les moutons dans la cage.

```
POUR CHASSE
   DONNE.VAR
   DEMANDE :TORTUE [INIT.ÉCRAN]
   DONNE.DÉMONS
DÉBUT
   SURVEILLANCE
FIN
POUR DONNE.VAR
   FAIRE « GRILLAGE 0
   FAIRE « MOUTON 0
   FAIRE « MOUTON1 3
   FAIRE « MOUTON2 2
   FAIRE « CHIEN 1
   FAIRE « VERT 92
   FAIRE « ORANGE 23
   FAIRE « NOIR 0
   FAIRE « BLANC 7
FIN
POUR INIT.ÉCRAN
   COLLISION
   FORMEGRILLAGE
   DONNECOULEURFOND :VERT
   CACHETORTUE
   LÈVEPLUME
   POSITIONNE [-150 -80]
   POSEPLUME
   DONNECOULEURCRAYON 0/:MARRON
   RECTANGLE 160 300
   LÈVEPLUME
FIN

POUR RECTANGLE :CÔTÉ1 :CÔTÉ2
   RÉPÈTE 2 [AV :CÔTÉ1 DR 90 AV :CÔTÉ2
   DR 90]
FIN

POUR DONNE.DÉMONS
   LORSQUE AU.DELÀ :MOUTON1 :GRILLAGE
   [DONNEVITESSE 0]
   LORSQUE AU.DELÀ :MOUTON2 :GRILLAGE
   [DONNEVITESSE 0]
   LORSQUE TOUCHER :MOUTON1 :MOUTON2
   [DONNEVITESSE 0]
   LORSQUE TOUCHER :CHIEN :MOUTON1]
   [DONNEVITESSE 0]
   LORSQUE TOUCHER :CHIEN :MOUTON2
   [DONNEVITESSE 0]
   LORSQUE 3 [DONNEVITESSE 0]
   LORSQUE 15 [DIRECTIONMANETTE]
FIN

POUR DIRECTIONMANETTE
   SI (MAN 1) < 0[STOP]
   DEMANDE :CHIEN [DONNEDIRECTION 45 * MAN 1]
FIN

POUR DÉBUT
   DONNE :MOUTON1 [-150 20] 45 :BLANC
   DONNE :MOUTON2 1 [150 20] 315:
   BLANC
   DONNE CHIEN 2 [0 0] 0 :NOIR
   DONNE.VITESSES
FIN

POUR DONNE :NO :FORME :POS :DIR:
   COULEUR
   DIRE :NO
   LÈVEPLUME
   DONNEFORME :FORME
   DONNECOULEUR :COULEUR
   DONNETORTUE
   DONNEPOS :POS
   DONNEDIR :DIR
FIN
POUR DONNE.VITESSES
   DEMANDE :MOUTON1 [DONNEVITESSE 10]
   DEMANDE :MOUTON2 [DONNEVITESSE 10]
   DEMANDE :CHIEN [DONNEVITESSE 60]
FIN
POUR SURVEILLANCE
   SI VITESSE = 0 [DÉTECTION]
   SURVEILLANCE
FIN
POUR DÉTECTION
   SI CONDITION AU.DELÀ :MOUTON1:
   GRILLAGE [DEMANDE :MOUTON1
   [AR 20 DR 90]]
   SI CONDITION AU.DELÀ :MOUTON2:
   GRILLAGE [DEMANDE :MOUTON2
   [AR 20 DR 90]]
   SI CONDITION TOUCHE :MOUTON1:
   MOUTON2 [HEURT]
   SI CONDITION TOUCHE :CHIEN:
   MOUTON1 [DEMANDE :MOUTON1
   [DR 90]]
   SI CONDITION TOUCHE :CHIEN:
   MOUTON2 [DEMANDE :MOUTON2
   [DR 90]]
   SI CONDITION3 [DEMANDE :TORTUE
   [TRACE.CAGE]]
   DONNE.VITESSES
FIN
POUR HEURT
   DEMANDE :MOUTON1 [DONNEDIRECTION AU.HASARD 360]
FIN
POUR DESSINE.CAGE
   LÈVEPLUME
   POSITIONNE [-150 -30]
   POSEPLUME
   DONNEDIR 90
   RÉPÈTE 2 [AV 50 DR 90]
   LÈVEPLUME
FIN
```

Exercices Logo

1. Modidifiez le jeu du chien de berger pour contrôler son déplacement au clavier.
2. Écrivez un programme de jeu de vaisseau spatial. Des météorites surgissent autour de vous, et vous devez les éviter et survivre aussi longtemps que possible. Voici quelques conseils. Utilisez une figure graphique pour le vaisseau et une autre pour les météorites. Utilisez les démons LORSQUE pour éviter les collisions. Les météorites se déplacent à vitesse constante mais selon une direction aléatoire (AU HASARD). Le vaisseau spatial peut être contrôlé avec les manettes.

Quatre à quatre

Après l'énorme succès du 64, le Plus/4 de Commodore représente un grand pas en avant : un BASIC bien supérieur, quatre logiciels intégrés, et 64 K de RAM directement accessibles par l'utilisateur.

Commodore prend bien soin de préciser que le Plus/4 n'a pas pour but de remplacer le 64; mais il a tant de possibilités supplémentaires qu'il pourrait bien éclipser complètement son prédécesseur.

Le Plus/4 est construit autour du microprocesseur 7501, qui dérive du 6502. Cette puce est conçue de façon à pouvoir manipuler plus de 64 K d'espace mémoire. L'usager aura donc toute la RAM à sa disposition et bénéficiera en même temps d'un BASIC décent. De fait, un programme BASIC peut atteindre jusqu'à 64 K (50 seulement s'il est fait usage de graphismes).

Le Plus/4 est de surcroît doté d'une excellente version du BASIC Microsoft, en particulier au niveau des commandes graphiques et sonores. DRAW trace des points et des lignes; PAINT remplit de couleurs des formes; BOX crée des carrés et des rectangles, que ce soient des blocs de couleur ou de simples contours; CIRCLE est d'un usage extrêmement souple. On peut tracer des cercles, mais aussi des ovales (en spécifiant la hauteur et la largeur), et même des arcs, en précisant simplement le point de départ et le point d'arrivée.

Toutes ces commandes fonctionnent en mode graphique, avec une résolution de 320 × 200 pixels. C'est la même que celle du 64, mais le Plus/4 a un nombre de couleurs infiniment plus élevé : 120 (plus le noir) d'entre elles peuvent être affichées simultanément. Il s'agit en fait de 15 teintes de base, dont chacune possède 8 niveaux de luminosité. Un gros regret toutefois : le Plus/4 n'a pas de lutins ce qui ne manque pas d'étonner pour une machine aussi récente.

Un basic enrichi

Le contrôle du son est assuré par le biais de commandes tout à fait familières. SOUND joue une note dont il faut spécifier la hauteur et la durée. VOL attribue au volume l'une des huit valeurs possibles sur un des deux canaux existant sur l'appareil, bien que le BASIC permette d'en manipuler trois : le « troisième » est en fait un générateur de bruit blanc, ce qui est très utile dans les jeux. Les sons produits sont émis à travers le haut-parleur du téléviseur ou du moniteur.

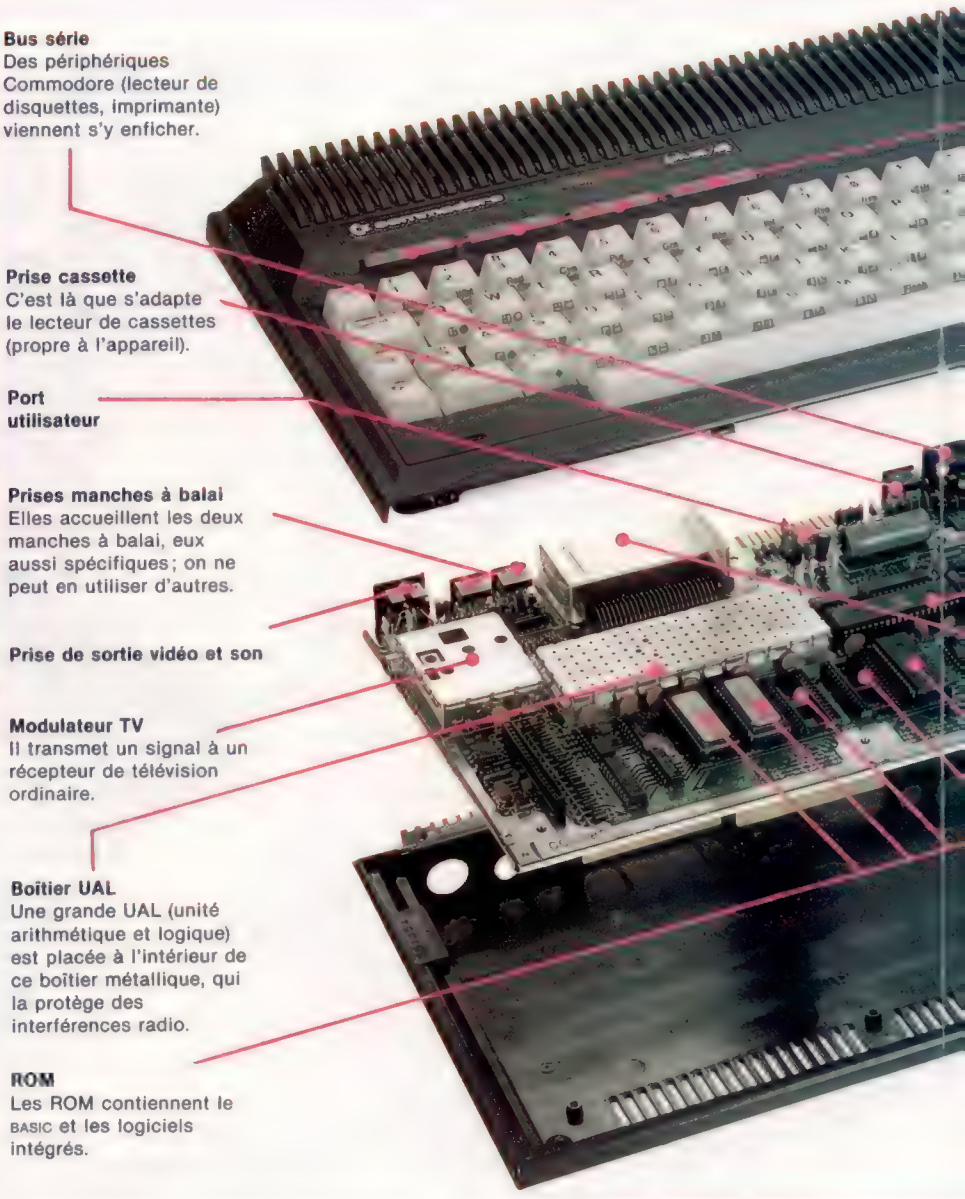
Le BASIC proprement dit a été enrichi de nombreuses fonctions très utiles. AUTO crée automatiquement de nouveaux numéros de ligne lorsqu'on tape un programme; RENUMBER leur donne ensuite des valeurs nouvelles et VERIFY permet de s'assurer que la sauvegarde (sur cassette ou sur disquette) s'est opérée correctement. De nombreuses commandes contrôlent le lecteur de disquettes. Le Plus/4 affiche 25 lignes de 40 caractères. Il est possible de choisir deux points sur l'écran et d'en faire les coins d'une « fenêtre ». Tout ce qui est texte (listages, commandes) n'apparaîtra qu'à l'intérieur de cette fenêtre, sans que le reste de l'affichage ne soit modifié.

Les touches du Plus/4 sont très sensibles, et une légère pression suffit. Certains caractères (@ , =, + ou −) se sont vu attribuer une touche séparée, et on peut également accéder aux caractères graphiques de façon directe. Quatre touches de fonction sont installées en haut du clavier; une fois l'ordinateur placé sous tension, elles sont automatiquement activées, de façon à produire les commandes les plus usitées. Il est par exemple tout à fait envisageable de créer une nouvelle commande comportant jusqu'à 128 caractères, et ce pour une seule touche. Grâce à l'emploi de la touche SHIFT, les touches de fonction peuvent donc assurer jusqu'à huit rôles différents.

Le vaste espace mémoire du Plus/4 permet d'accueillir quatre logiciels intégrés : un traitement de texte, un tableur, une base de données,

L'avenir?
S'il s'est fait longtemps attendre, le successeur du Commodore 64 a de nombreuses caractéristiques propres aux micro-ordinateurs les plus récents : conception de type MSX, pavé curseur, vaste mémoire et logiciels intégrés. Mais la concurrence est extrêmement forte, et l'acheteur potentiel mieux informé que jamais.
(Cl. Chris Stevens.)

Le QL face au Plus/4
Objectivement, la comparaison semble sans objet : le QL dispose de Microdrives, de bons logiciels intégrés, d'une mémoire plus vaste et du superBASIC, le tout pour 6 000 F environ ; pour un prix à peine inférieur (lecteur de cassettes compris), le Plus/4 ne peut guère faire valoir que son clavier. Mais l'achat d'un appareil se fait rarement sur des critères aussi techniques, et le consommateur choisit souvent au sentiment. (Cl. Ian McKinnell.)



un programme graphique. Tous ces logiciels sont censés pouvoir travailler ensemble.

Malheureusement, le traitement de texte n'est pas d'un emploi très facile. Le Plus/4 ne peut afficher simultanément que 40 caractères, alors que la plupart des imprimantes ont une capacité double. L'écran se déplace donc dès que l'on a atteint la trente-septième lettre, et ce jusqu'à la soixante-dix-septième ; il revient alors à son point de départ. Certaines commandes de mise en forme permettent de fixer les marges et de justifier le texte, mais leur effet n'est visible qu'à l'impression. SEARCH et REPLACE retrouvent certains mots ou certaines phrases dans le corps du document et les remplacent au besoin par quelque chose d'autre. Malheureusement, il n'est pas possible d'entrer plus de 99 lignes de texte.

Si le tableur est de maniement plus aisé, il souffre lui aussi d'un affichage limité à 40 colonnes. En effet, il ne peut présenter à la fois que trois cases en largeur et douze en profondeur, bien qu'il puisse gérer des modèles de 17 cases sur 50.

Des surprises

Le logiciel graphique est très décevant. Il se borne, à partir des données fournies par le tableur, à créer des graphiques assez sommaires réalisés avec de simples blocs, et à les transmettre au programme de traitement de texte. Ils peuvent être imprimés ou simplement affichés. Un résultat encore surprenant et bien décevant.

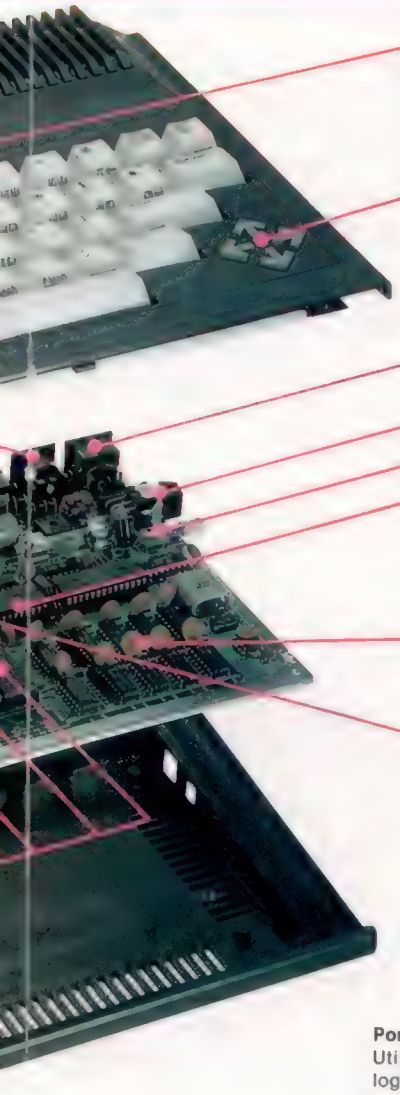
Bien que le traitement de texte et le tableur puissent être mis en œuvre directement, on ne peut sauvegarder les résultats obtenus que sur disquette. Cela signifie qu'il y a peu de possibilités pour les utilisateurs de mémoires à cassette.

Le dernier programme, la base de données, ne peut fonctionner sans ce périphérique. Un format standard est défini, puis stocké sur disquette, sous forme de fichier « vide », au sein duquel on entre ensuite les données nécessaires. Cela veut dire qu'une disquette ne peut servir que pour une seule base de données, et surtout qu'il n'est pas possible de modifier le format général une fois les informations stockées. Chaque fichier peut contenir jusqu'à 999 fiches, dont chacune renferme un maximum de 17 rubriques de 38 caractères.

Dans l'ensemble, il faut bien dire que le logiciel n'est pas à la hauteur de ce que l'on pouvait espérer pour une telle machine. Ils sont trop primaires pour être utilisés en gestion, et il est indispensable, pour en faire usage, d'acquérir un lecteur de disquettes, d'où des frais supplémentaires.

L'utilisateur, en revanche, appréciera sans doute vivement Tedmon, le moniteur langage machine, qui peut être mis en route par la commande MONITOR.

Commodore a aussi prévu de nombreux périphériques pour le Plus/4. Le plus important est sans doute le lecteur de cassettes, qui, dans la bonne tradition de la firme, est spécifique à l'appareil ! En outre, il est impossible d'employer les lecteurs du Vic-20 ou du 64, la prise n'étant



Touches de fonction
Chacune peut être programmée de façon à produire deux commandes.

Pavé curseur

Prise d'alimentation

Commutateur on/off

Bouton de remise à zéro

UC 7501

RAM de 64 K

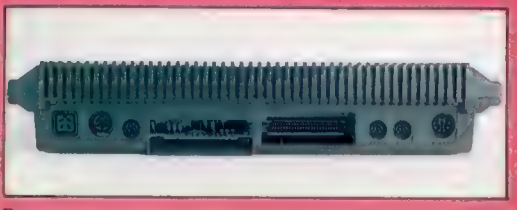
Port d'extension
Utilisé par les logiciels sur cartouche et par un lecteur de disquettes « rapide ».

Touches de fonction

Touche curseur

Bouton de remise à zéro

Port d'extension et utilisateur



Du nouveau :
Les vieux amateurs de Commodore apprendront avec intérêt que le Plus/4 est doté de touches ESCAPE et RESET, d'un pavé curseur, de touches de fonction programmables à partir du BASIC et d'une touche HELP. Toutefois, la prise d'alimentation, les prises cassettes et manches à balai, les ports d'extensions et utilisateurs sont tous incompatibles avec ceux du Vic-20 et du Commodore 64. (Cl. Chris Stevens.)

Commodore Plus/4

PRIX
★ ★ ★

DIMENSIONS
67 × 203 × 338 mm.

UC
7501, à 0,9 ou 1,8 MHz.

MÉMOIRE
64 K de RAM, 64 K de ROM.

ÉCRAN
Texte : 25 lignes de 40 caractères. Graphique : 320 × 200 en 121 couleurs.

INTERFACES
2 prises manches à balai, interface série, interface cassette, port parallèle et cartouche.

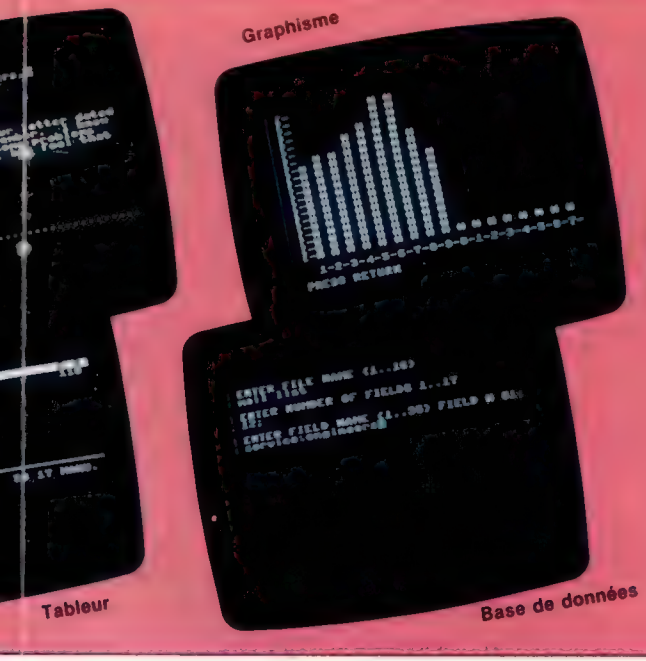
LANGAGE DISPONIBLE
BASIC.

CLAVIER
Type machine à écrire, 67 touches dont 4 touches de fonction.

DOCUMENTATION
Des manuels décents (mais pas toujours très bien rédigés) expliquent comment programmer et se servir des logiciels intégrés.

POINTS FORTS
Très bon BASIC (graphisme, programmation structurée). 64 K de RAM réellement accessibles à l'utilisateur.

POINTS FAIBLES
Nécessité d'un lecteur de cassette spécifique, manches à balai non standards. Les logiciels intégrés sont assez médiocres et difficiles à mettre en œuvre sans lecteur de disquettes.



Des frais supplémentaires

Le Plus/4 comporte quatre logiciels intégrés : un traitement de texte, un tableur, une base de données et un utilitaire graphique ; mais ils ne fonctionnent réellement qu'avec un lecteur de disquettes, ce qui accroît de façon importante le prix réel de l'appareil. (Cl. Ian McKinnell.)

pas la même ! Et c'est aussi le cas pour les manches à balai...

Le Plus/4 peut fonctionner avec une variante, légèrement modifiée, du lecteur de disquettes 1541, bien connu pour sa lenteur. Commodore a promis une nouvelle version, plus rapide, qui s'enficherait dans le port cartouche. Par ailleurs la firme propose cinq imprimantes différentes : une à marguerite, trois matricielles (dont une en couleur) et une imprimante-table traçante à quatre stylos à bille de couleur. Mais ce n'est encore qu'une proposition pour l'avenir... proche toutefois.

Si le Plus/4 est vendu à un prix un peu plus élevé que le 64, un BASIC bien supérieur et un espace mémoire plus vaste en font toutefois un achat très intéressant. Nous consacrerons ultérieurement un article au fonctionnement du son BASIC. Bien entendu, il lui faudra se tailler une place sur le marché avant qu'apparaissent des programmes qui lui seront consacrés ; mais vu la place prépondérante de Commodore dans le domaine de la micro-informatique individuelle, cela ne devrait pas être difficile.

Envoyez les clones

Nous allons étudier les caractéristiques de Vu-Calc, un progiciel tableur et générateur de modèles financiers, de remboursement d'emprunts et de prêts bancaires.

Le point fort des programmes tableurs, même avec un simple progiciel comme Vu-Calc de Psion, est de pouvoir appliquer des formules mathématiques complexes aux données. Comme nous allons le voir, il est possible d'établir des modèles de calcul intéressants et très utiles avec Vu-Calc, bien que ce programme ne prévoie pratiquement rien comme formules mathématiques intégrées, à l'exception de la somme d'ensembles de positions (que l'on indique en préfixant l'adresse de la case, du signe ◊ ou @).

Les tableurs plus sophistiqués comportent des formules mathématiques poussées que l'utilisateur peut appeler en utilisant leur nom. L'avantage d'un tel système est que vous n'avez pas à vous préoccuper de la manière dont les calculs sont effectués. Si vous voulez utiliser une formule mathématique de calcul des mensualités de remboursement d'un emprunt immobilier avec Multiplan par exemple, sur plusieurs périodes de temps, disons quinze, vingt et vingt-cinq ans, vous devez seulement appeler la formule et saisir les données appropriées. Multiplan se charge de trouver les réponses.

Avec Vu-Calc, les mêmes calculs prendront beaucoup plus de temps et d'efforts. Vous devrez établir vous-même les formules voulues et les entrer dans la machine. Vu-Calc impose à l'utilisateur un certain nombre de contraintes. Il comporte un maximum de vingt-huit colonnes, ce qui signifie que le plus grand modèle que vous pouvez faire, avec une colonne par mois, portera seulement sur une période de deux ans. Un problème de précision des calculs se posera également : Vu-Calc, ne travaillant qu'avec des nombres entiers, ignore les chiffres après la virgule. Pour lui, 99,5 est égal à 99! Il autorise la saisie de valeurs et d'opérations arithmétiques en tous points du modèle. Par exemple, lorsque le curseur est situé sur une case vide (prenons H5), vous pouvez taper 500*2 sur la ligne de commande. La touche d'exécution affiche le résultat, 1000, à la case H5.

Une autre tracasserie, propre à Vu-Calc, concerne l'édition des formules. Un bon progiciel comme Lotus 1-2-3 utilise une touche de fonction pour l'édition. Sa frappe positionne automatiquement le contenu de la case du curseur sur la ligne de commande. Vu-Calc comporte une commande ÉDIT (◊E) à utiliser lorsqu'une formule doit être modifiée, mais la formule doit être retapée à chaque nouvel usage de la commande d'édition. Si vous travaillez sur une formule assez longue et que vous apercevez une parenthèse oubliée, il ne vous sera pas possible de l'insérer. Toute la ligne devra être retapée. L'action de EDIT se limite à effacer l'ancienne formule et à la remplacer par la nouvelle.

La commande REPLICATE (◊R) associée à une formule permet de créer des modèles assez complexes. Supposons que vous vouliez approfondir l'application « budget familial », pour évaluer à l'avance l'effet de l'inflation sur le poste « nourriture », en vous basant sur un taux d'inflation de 0,5 % par mois. Le calcul à la main prendrait manifestement beaucoup de temps. Avec Vu-Calc, cela se fait très rapidement par l'intermédiaire de la commande REPLICATE.

Pour mener à bien l'opération, vous demandez à Vu-Calc d'augmenter votre budget initial mensuel (par exemple de 2 000 F), de 0,5 %. Vu-Calc suppose que l'on saisisse les opérations arithmétiques à effectuer (on utilise la commande GROWBY avec des tableurs plus élaborés). Afin qu'il soit à même de reconnaître une formule comportant des adresses de positions, la formule doit être précédée soit de « $ », soit de « % ». Il s'agit de deux signes arbitraires qui n'ont rien à voir avec les dollars ni avec les pourcentages, mais qui indiquent que les adresses de la formule doivent être prises en considération et qu'elles sont soit relatives (%), soit absolues ($). Une référence absolue à une position ordonne au programme d'intervenir sur une valeur d'une case spécifique, sans tenir compte de sa position.

Revenons à notre exemple, pour voir à quoi correspond une position relative. La formule, pour accroître le budget de 0,5 %, est %B3*100.5/100, avec % pour indiquer l'adresse relative de la position, et B3 pour l'adresse de la valeur du poste mensuel « nourriture ». Ayant mis cette formule dans la case B4, nous devons encore la reproduire sur l'ensemble du tableau pour toute l'année. B4 affiche le résultat numérique de la formule, celle-ci apparaissant en bas du document lorsque le curseur est en B4. La commande REPLICATE :◊R,B4,B5:B14 donne le résultat voulu (B4 contient la formule, B5:B14 définit l'ensemble des cellules concernées). Les résultats sont affichés presque instantanément; notre modèle de tableau électronique ayant alors l'aspect suivant :

	1	2	3	4	5
A			JAN	FÉV	MAR
B	NOURRITURE		2000	2010	2020

L'affichage montre pourquoi B3 est utilisé comme valeur mensuelle initiale du budget; l'appellation porte sur les colonnes 1 et 2, et nous commençons en colonne 3 pour une meilleure lisibilité du tableau. Vous remarquez que toutes les valeurs sont des nombres entiers — la colonne MARS devrait contenir la valeur 2020,50, mais le tableur arrondit à la décimale inférieure, ce qui affiche 2020 tout rond. Le chiffre pour AVRIL est en réalité de 2030,1502, Vu-Calc le résultat à 2030. Comme l'accroissement dû à l'inflation est davantage prononcé d'un mois sur l'autre, l'écart entre la valeur réelle et la valeur affichée augmente de plus en plus.

Ce simple exemple montre l'effet de la commande REPLICATE lorsqu'elle est utilisée sur des adresses relatives. Chaque fois que le programme réécrit la formule une position plus à droite, la formule est modifiée en conséquence. Notre formule originelle en B4 était %B3*100.5/100. La formule est recopiée en B5 comme %B4*100.5/100; en B6 comme %B5*100.5/100, etc. Chaque fois, le numéro de colonne de l'adresse est augmenté d'une valeur. La recopie d'une formule en descendant une colonne a le même effet sur les adresses (E1 devient F1, etc.). Si nous avions utilisé des adresses absolues au lieu d'adresses relatives, ce décalage ne se serait pas présenté. La même formule aurait été recopiée à toutes les positions, et la valeur de chacune identique à celle de B4.

Essayons d'appliquer maintenant le même modèle à l'estimation du coût mensuel d'approvisionnement en matières premières d'une société. La valeur d'origine est de 100 000 F par mois, et l'accroissement est de 0,5 % par mois sur deux ans. Quel coût supplémentaire représenterait l'achat des produits en milieu de deuxième année? Le modèle nous permet d'effectuer ce calcul très rapidement.

Transformez en 100 000 la valeur en B3, en déplaçant le curseur en B3 et en entrant le nouveau chiffre. Utilisez maintenant la commande REPLICATE pour étendre la formule aux positions B14 à B26, sur toute l'année. Avec Vu-Calc, vous devez recalculer les résultats (qui sont alors toujours fondés sur l'ancienne formule), en utilisant la commande CALCULATE, ♯C. Vu-Calc calcule alors les nouvelles valeurs et affiche le résultat voulu en B20. Si vous appliquez cet exemple, vous obtiendrez 109 931 F, soit une augmentation de pratiquement 10 000 F. Il ne s'agit pas d'une valeur précise, puisque tous les chiffres sont arrondis, mais elle est assez représentative pour donner une idée de l'effet de l'inflation sur cette période.

Comme dernier exemple d'un problème portant sur une seule ligne de tableau, envisageons une formule de modèle financier plus complexe. Il s'agit de calculer le solde en voie de diminution d'une dette de 1 000 F sur une carte de crédit ou sur un emprunt bancaire, avec des intérêts de 27 % par annuité. En supposant que 80 F soient remboursés chaque mois, au bout de combien de temps la dette sera-t-elle apurée? Les informations nécessaires pour ce calcul sont le montant principal de l'emprunt, le taux d'intérêt pour un mois et le remboursement mensuel. Si nous entrons la valeur 1 000 en B1, la formule sera %B1+%B1*.27/12-80.

Recopiez la formule sur les 28 colonnes du modèle, passez en revue la rangée pour voir à quel moment la valeur devient positive, et vous saurez à partir de quand la dette sera remboursée et quand vous deviendrez créditeur si les versements continuaient.

Selon notre modèle, il faudrait seize mois. Le tableau permet également de voir la balance mensuelle selon le même remboursement.

Relativement absolu

Dans ce modèle simple, nous avons utilisé la commande REPLICATE sur les lignes C, D et E. Nous avons écrit, pour la clarté de l'exemple, la formule pour chaque position. C3 a été reproduit de manière absolue sur toute la ligne; la même formule A5/12 apparaît donc sur toutes les cases C, avec le même résultat de 450 F. En revanche, les formules en D4 et E3 ont été recopiées de manière relative. Les références de la position dans la formule changent donc tout au long de la ligne, ainsi que les résultats afférents. (Cl. Liz Dixon.)

	1	2	3	4	5
A			REVENU ANNUEL = 5400		
B			JAN	FÉV	MAR
C	REVENUS DU MÉNAGE		A5/12 450	A5/12 450	A5/12 450
D	DÉPENSES DU MÉNAGE		DATA 400	D3*1.01 404	D4*1.01 408
E	BALANCE MENSUELLE		C3-D3 50	C4-D4 46	C5-D5 42

Contrôler la puissance

La construction d'un convertisseur numérique/analogique va nous permettre de commander des dispositifs analogiques à partir du port utilisateur et de produire un son synthétisé numériquement.

Pour ce projet, nous avons choisi d'utiliser une puce convertisseur numérique/analogique (N/A) prête à l'emploi, bien qu'il soit possible de construire un circuit avec des composants discrets.

La sortie analogique provenant de cette puce est shuntée par un amplificateur placé sur une seconde puce. La sortie qui en résulte est reliée directement à l'une des sorties, et par un condensateur et un compteur à l'autre.

Étape 1. Coupez le boîtier pour placer les deux connexions du bus du système. Une prise peut aussi être coupée pour des projets futurs.

Étape 2. Coupez la carte de montage de manière à avoir 16 bandes de 30 trous. Faites les ruptures de pistes comme sur le diagramme. Soudez d'abord les prises de la puce, puis les liaisons des fils. Les deux condensateurs doivent ensuite être installés. Le sens n'a pas d'importance. Si vous désirez installer la prise d'extension de bus, soudez-la d'abord, et installez le câble-ruban.

Étape 3. Placez les quatre prises dans le boîtier, avec le potentiomètre. Établissez les connexions entre celles-ci et le fil d'étain. Enfin, posez les trois conducteurs sur la carte.

Étape 4. Enfichez les deux puces, et le convertisseur est complet. Notez que les deux puces ne s'enfichent pas dans le même sens : le convertisseur N/A doit être positionné de telle sorte que l'encoche se trouve à gauche, vue du haut, le connecteur de bus mâles étant vu vers le haut, l'encoche de la puce amplifiant doit être à droite.

Après avoir construit le convertisseur N/A et avoir soigneusement vérifié toutes les connexions, vous pouvez tester l'unité. Ce convertisseur transformera toute valeur binaire à 8 bits placée dans le registre de données du port utilisateur en une tension. Cette tension sort de l'unité de deux manières.

Au niveau de la paire de prises de sortie CC (courant continu), une tension CC de 0 à 2,5 V est disponible, correspondant aux valeurs numériques du port utilisateur de 0 à 255. L'autre paire de sorties nous permet de simuler une sortie CA (courant alternatif).

Le niveau de tension est commandé par un potentiomètre et peut être ajusté pour correspondre à l'entrée requise par un autre équipement.

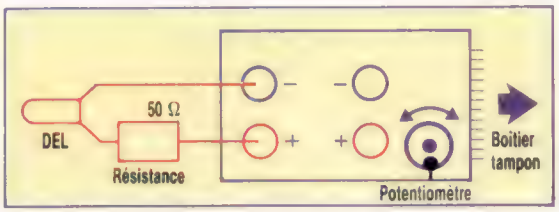
Pour tester l'unité, nous pouvons concevoir une expérience simple, qui modifie l'intensité d'une DEL (diode électroluminescente). Pour cela, suivez ces directives :

Étape 1. Une DEL, du type utilisé dans le boîtier tampon initial, doit être connectée en série à une résistance de 50 Ω .

Obtenir des nombres
Ce convertisseur numérique/analogique montre un potentiomètre beaucoup trop important. Ce n'est pas nécessaire. Il faut savoir, avant d'acheter des composants électroniques, ce à quoi ils vont servir.
(Cl. Ian McKinnell.)

Étape 2. Le convertisseur N/A doit être connecté directement au boîtier tampon, lui-même connecté au port utilisateur et alimenté de la manière habituelle.

Étape 3. La DEL et la résistance doivent être connectées aux prises de sortie CC sur le convertisseur; et ce programme doit être exécuté :



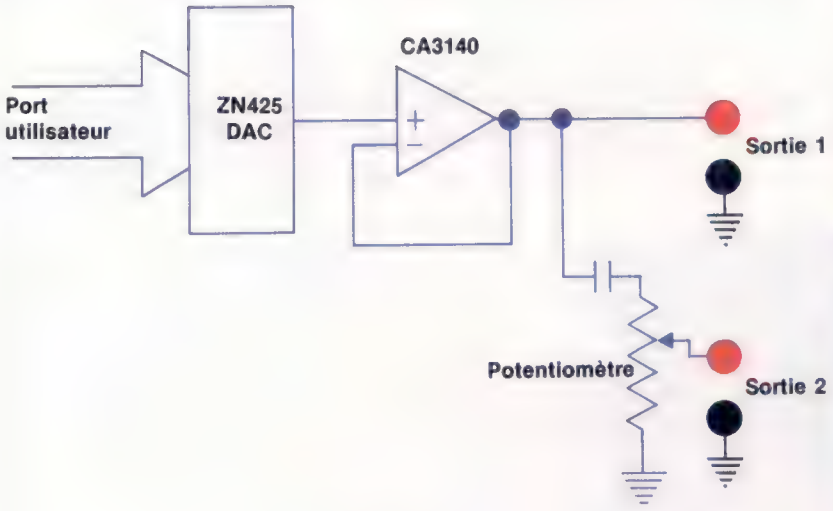

```
10 REM **** PROGRAMME TEST CBM 64 N.A ****
20 DDR=56579:DATREG=56577
30 VL=127
40 POKE DDR,255:REM TOUTES SORTIES
50 POKE DATREG,VL
60 PRINT VL
70 GET A$
80 IFA$<>"Z" AND A$<>"X" THEN 70
90 IF A$="X" THEN DV=1
100 IF A$="Z" THEN DV=-1
110 VL=VL+DV
120 IF VL<256 AND VL>=0 THEN 50
```

```
10 REM **** PROGRAMME TEST BBC N.A. ****
20 DDR=&FE62:DATREG=&FE60
25 value=127
30 ?DDR=255:REM TOUTES SORTIES
40 REPEAT
55 ?DATREG=VALEUR
57 PRINT VALEUR
60 A$=GET$
62 IF A$<>"Z" AND A$<>"X" THEN 60
65 IFA$="X" THEN dv=1 ELSE dv=-1
67 value=VALEUR+dv
70 UNTIL(value>255 OR VALEUR<0)
```

Ce simple programme affecte les huit lignes du port utilisateur à la sortie en plaçant 255 (c'est-à-dire 11111111) dans le registre de direction des données. Une valeur initiale de 127 est alors placée dans le registre de données du port utilisateur. En appuyant sur les touches Z et X, la valeur présente dans le registre de données diminue ou augmente de façon correspondante. Le programme se termine lorsque la valeur du registre de données se situe au-delà de la plage 0 à 255.

En augmentant la valeur numérique présente dans le registre de données, nous pouvons produire des tensions analogiques croissantes pour alimenter la DEL. Lorsque la tension atteint un niveau acceptable, la DEL commence à briller, la brillance maximale étant obtenue lorsque la valeur de 255 est présente dans le registre de données du port utilisateur.

Circuit complet

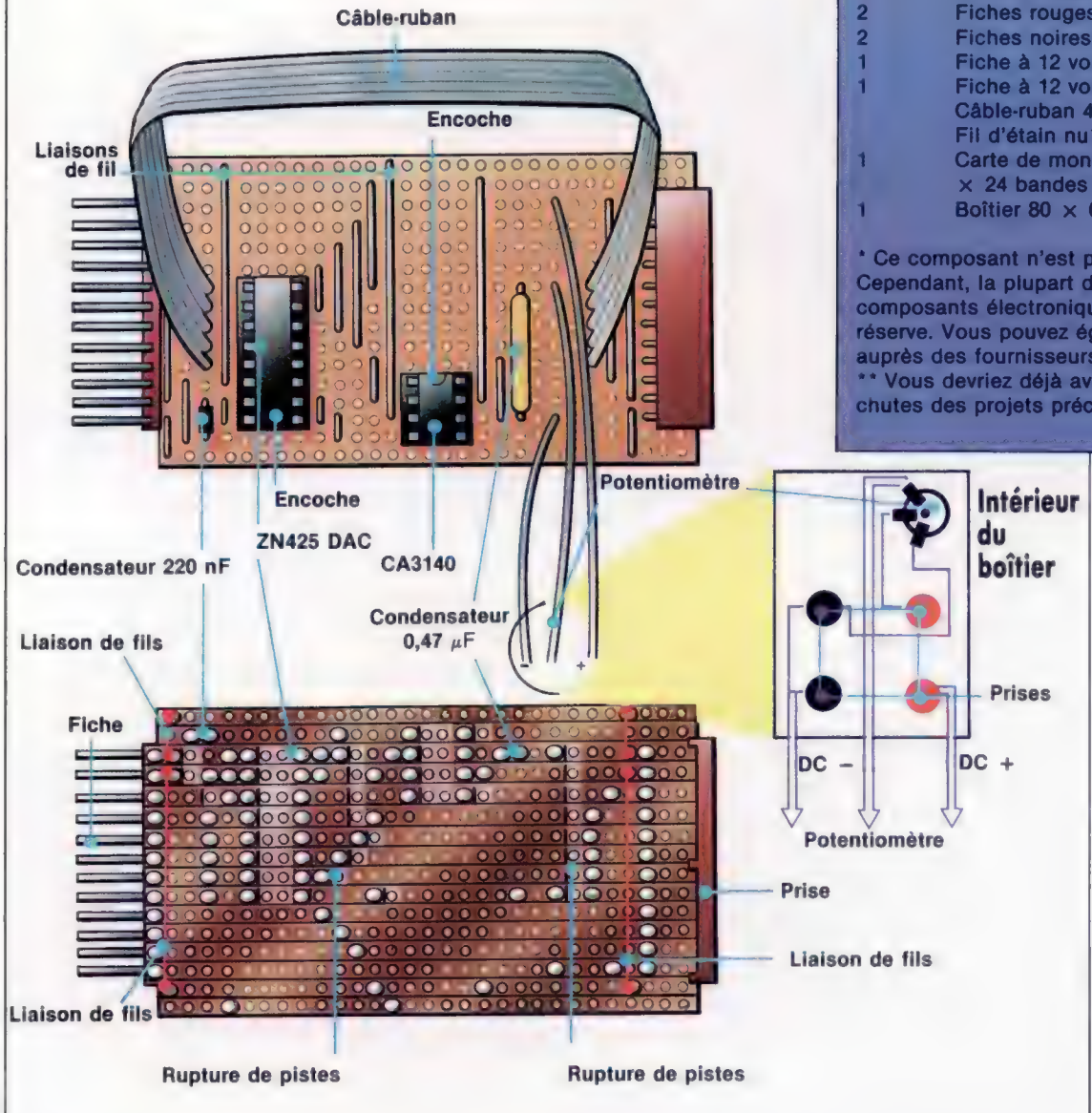


Comme d'habitude dans ce type de construction, vous devez soigner les détails, surtout l'emplacement et la netteté des ruptures de pistes. Vérifiez votre travail avec le voltmètre; insérez d'abord les composants passifs et les liaisons de fil; soignez les soudures; veillez à placer correctement les puces.

Liste des pièces

Quantité	Article	N° Maplin
1	ZN425 DAC*	
1	Amplificateur	QH29G
1	Prise DIL à 16 touches	BL19V
1	Prise DIL à 8 broches	BL17T
1	Condensateur 220 nF	WW83E
1	Condensateur 0,47 µF	BX80B
1	Potentiomètre 10 kΩ	FW02C
2	Prises rouges 4 mm	HF73Q
2	Prises noires 4 mm	HF69A
2	Fiches rouges 4 mm	HF66W
2	Fiches noires 4 mm	HF62S
1	Fiche à 12 voies	YW19V
1	Fiche à 12 voies	YW30H
	Câble-ruban 4 pouces 5 voies**	
	Fil d'étain nu**	
1	Carte de montage 50 trous × 24 bandes	FL07H
1	Boîtier 80 × 61 × 41 mm	LH20W

* Ce composant n'est pas offert par Maplin. Cependant, la plupart des boutiques de composants électroniques devraient l'avoir en réserve. Vous pouvez également le commander auprès des fournisseurs de composants.
** Vous devriez déjà avoir ces composants, chutes des projets précédents.



Kevin Jones

Atterrissage sur CM 64

Encore un jeu que Pierre Monsaut a écrit pour plusieurs micro-ordinateurs. Cette version est destinée au Commodore 64. Un conseil : enregistrez ce jeu sur cassette pour éviter des ennuis.

Après un long voyage en apesanteur, poser une navette spatiale en douceur n'est pas chose aisée ; mais grâce à votre ordinateur, vous allez être en mesure de vous entraîner sans danger. Vous devez poser votre navette sur l'aire prévue à cet effet. Vous pouvez vous diriger vers la droite et vers la gauche à l'aide des touches de contrôle du curseur ou freiner votre descente avec la barre d'espacement. Pour réussir un atterrissage, les vitesses verticales et horizontales doivent être inférieures ou égale à 1.

```
5 REM  xxxxxxxxxxxxxxxxx
10 REM x ATTERRISSAGE x
15 REM xxxxxxxxxxxxxxxxx
20 GOSUB 1000
100 GET X$:IF FU=0 THEN 150
110 IF X$<>"" THEN FU=FU-1
120 IF X$=" " THEN VV=VV-1
130 IF X$=G$ THEN VH=VH-1
140 IF X$=D$ THEN VH=VH+1
150 VV=VV+0.1:V=V+VV:H=H+VH
190 IF H>255 THEN H=0:MS=M1
200 IF H>90 AND MS=M1 THEN 4540
210 IF H<0 AND MS=M1 THEN H=255:MS=M0
220 IF H<0 AND MS=M0 THEN 4540
240 PRINT HO$;
250 PRINT "CARBURANT :";STR$(FU);"
255 IF V<0 THEN 300
260 POKE D+16,MS:POKE D,H
270 POKE D+1,INT(V)
275 IF V>V1-3 THEN 3000
300 PRINT "VITESSE VERT. :";STR$(INT(10*
VV)/10);"    "
310 PRINT "VITESSE HORIZ.:";STR$(VH);"
  "
320 GOTO 100
1000 D=53248:RESTORE
1010 FOR I=0 TO 190
1020 READ A:POKE 832+I,A
1035 NEXT I
1040 M1=5:M0=0:POKE D+39,7:POKE D+40,7
1090 POKE D+41,0:POKE 2040,13
1110 POKE 2041,14:POKE 2042,15
1130 V1=230:V0=0:FU=90:G$=CHR$(29)
1170 D$=CHR$(17):HO$=CHR$(19):M=54272
2000 PRINT CHR$(147);:POKE D+21,0
2020 H=INT(RND(TI)*191)+65
2030 AH=INT(RND(TI)*191)+65
2050 FU=FU+10:MS=M0:VH=0:VV=0:V=0
2100 POKE D+2,AH:POKE D+3,V1
2120 POKE D,H:POKE D+1,V:POKE D+21,3
2150 RETURN
3000 IF ABS(H-AH)>4 THEN 4000
3005 IF VV>1 THEN 4000
3010 IF ABS(VH)>1 THEN 4000
3020 FOR I=1 TO 4000:NEXT I
3030 SC=SC+1:GOSUB 2000
3050 GOTO 100
4000 POKE D+5,V+5:POKE D+4,H
4020 POKE D+21,6
4500 PRINT:PRINT:PRINT:PRINT:PRINT
4530 PRINT TAB(6)"VOTRE NAVETTE S'EST EC
RASEE"
4540 IF SC>RE THEN RE=SC
4545 FOR I=1 TO 2000:NEXT I
4560 PRINT:PRINT:PRINT
4580 PRINT TAB(13)"SCORE :";SC
4590 SC=0
5000 PRINT:PRINT:PRINT
5030 PRINT TAB(13)"RECORD :";RE
5040 IF RE<SC THEN RE=SC
5050 SC=0:PRINT:PRINT:PRINT:GET X$
5100 PRINT TAB(13)"UNE AUTRE ?"
5110 GET X$
5120 IF X$="" THEN 5110
5130 IF X$<>"N" THEN POKE D+21,0:GOTO 20
5140 END
10000 DATA 0,255,0,1,255,128,3,255,192
10010 DATA 7,255,224,12,195,48,12,195,48
10020 DATA 15,255,240,12,102,48,12,102,4
8
10030 DATA 15,255,240,7,255,224,3,129,19
2
10040 DATA 1,129,128,0,255,0,1,255,128
10050 DATA 1,24,128,2,60,64,2,36,64
10060 DATA 4,0,32,4,0,32,14,0,112,0
10100 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
10110 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
10120 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
10130 DATA 0,0,0,0,0,0,0,0,0,0,255,255,255
10140 DATA 255,255,255,255,255,255,0
10200 DATA 0,0,0,0,0,45,0,27,3,0,0,8,0,1
27,0
10210 DATA 0,5,0,0,236,0,0,67,0,2,64,240
10220 DATA 7,32,112,7,32,56,15,145,56
10230 DATA 31,23,120,31,25,124,95,97,254
10240 DATA 127,255,254,255,255,255,255
10250 DATA 255,255,0,0,0,0,0,0,0,0,0
```

Libre transfert

En approchant de la fin de notre cours, nous abordons des techniques plus générales : le code translatable, les longueurs d'instructions et les routines de synchronisation.

Un programme écrit à l'aide de code *translatable,* ou *indépendant de la position,* peut être placé en n'importe quelle position de mémoire et exécuté sans nécessiter aucune modification. C'est particulièrement important dans les systèmes multitâches ou multi-utilisateurs, où plusieurs programmes peuvent être chargés en mémoire en même temps, à l'endroit le plus approprié. Même dans des systèmes plus simples, à un utilisateur, il est généralement important de pouvoir disposer de bibliothèques de sous-programmes et de construire un programme à partir de modules autonomes, la position d'une routine en mémoire pouvant varier.

La plupart des processeurs règlent cette question à l'aide de ce qu'on appelle un *chargeur-éditeur de lien.* L'assembleur produit un code translatable, qui exclut toute référence à des adresses réelles en mémoire; c'est le rôle du chargeur-éditeur de lien d'insérer les adresses en chargeant le programme en mémoire. Puisque c'est le chargeur lui-même qui manie les adresses, il est simple de s'assurer que les transferts de contrôle entre différents modules sont correctement traités.

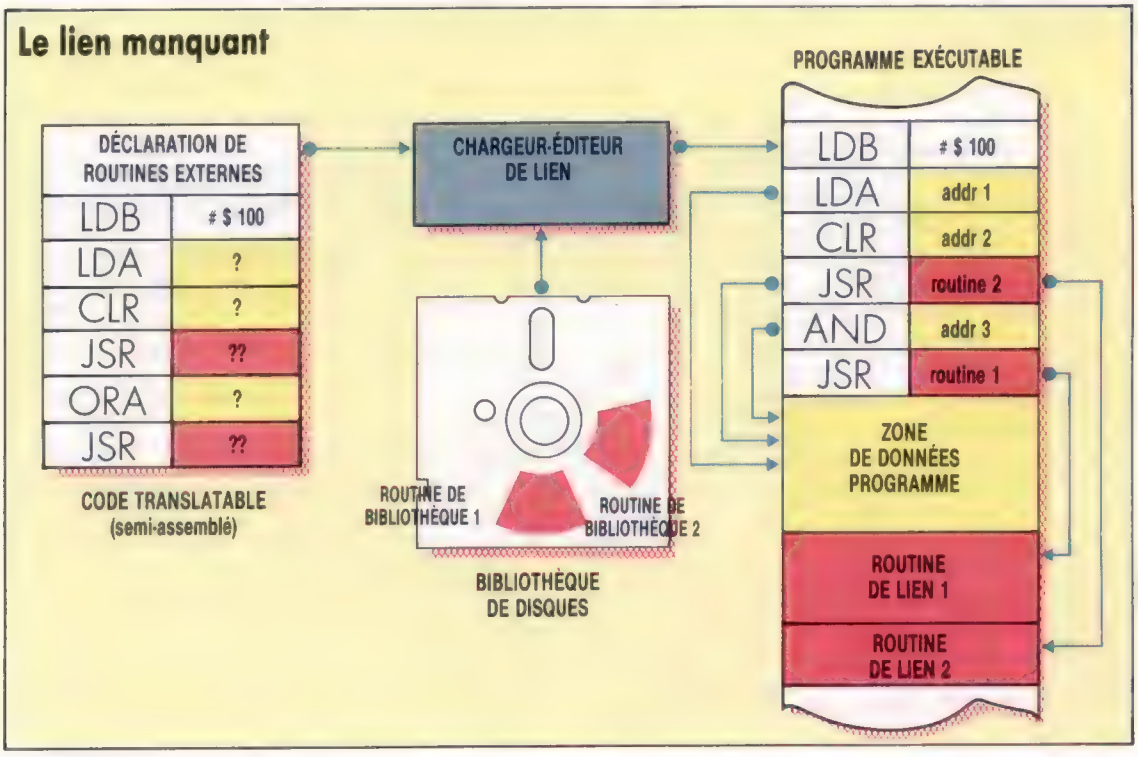
Ainsi, des parties de code peuvent être écrites en différents langages qui sont tous compilés ou assemblés pour former le même code translatable; par exemple, des programmes en PASCAL peuvent appeler des routines de bibliothèque en FORTRAN.

Cette approche peut aussi être utilisée avec le 6809, et, en fait, elle est nécessaire si l'on emploie une construction modulaire. Le 6809 facilite ce processus en autorisant à écrire directement un code translatable.

La clé pour écrire un code translatable consiste à se référer à toutes les adresses au moyen d'un *décalé* (offset) du compteur de programme (PC). Il existe pour un programme deux moyens d'utiliser une adresse : comme donnée et comme destination pour un transfert de contrôle.

Les instructions de branchement (BRA, BSR, etc.) calculent leurs destinations comme décalés du PC, et pourraient servir à tous les transferts de contrôle à l'intérieur du programme utilisateur. Les instructions de transfert absolu (JMP et JSR) ne devraient être utilisées que pour des destinations qui seront toujours au même endroit en mémoire, comme le sont les routines de système d'exploitation.

Le plus difficile est de rendre indépendantes toutes les références à une position de donnée, et le 6809 y arrive en permettant d'utiliser le PC



Chargeur-éditeur de lien
Dans de grands systèmes, les programmes en langage machine sont en fait chargés en mémoire par le chargeur-éditeur de lien. Cette utilité du système d'exploitation prend le code machine semi-assemblé (ne contenant pas d'adresses absolues) à partir de l'assembleur, et détermine la meilleure adresse ORG à partir de l'état du système. Il utilise cette adresse pour remplacer les adresses symboliques, que l'assembleur a laissées dans le programme, par des adresses absolues, puis lie au programme toutes les routines de bibliothèque nécessaire au programmeur; ces routines sont chargées à partir du disque de bibliothèque et attachées au programme. Leurs adresses d'appel absolues peuvent ensuite remplacer les adresses symboliques dans le programme. Enfin, le chargeur passe la totalité du programme au système d'exploitation pour l'exécution.
(Cl. Liz Dixon.)

pour l'indexation. L'instruction :

```
LDA OFFSET,PC
```

additionnera le décalé (signé) à la valeur actuelle du PC pour obtenir l'adresse effective. Le problème avec ce mode d'adressage est de savoir comment calculer correctement le décalé : il faut calculer la différence entre l'adresse de la donnée et la valeur actuelle du PC, en se souvenant que le PC est incrémenté dès qu'une instruction est chargée dans le processeur. Lorsque l'on exécute l'instruction, le PC pointe donc l'instruction suivante.

Cette méthode est compliquée par l'ampleur des variations de longueur des instructions 6809 — de 1 à 5 octets de longueur. Par exemple :

```
LDX OFFSET,Y
```

prend 1 octet pour l'opc et 1 octet pour l'après-octet, qui est utilisé pour toute instruction indexée pour spécifier le registre d'index qui est pris en compte. Le décalé peut prendre 1, 2 ou 3 octets, suivant sa taille. Les décalés nuls et des décalés exprimés en 5 bits peuvent être incorporés dans l'après-octet (bien que certains assembleurs ne puissent pas manier cela très précisément). Les décalés supérieurs requièrent un octet supplémentaire (s'ils peuvent s'exprimer en 8 bits) ou 2 octets supplémentaires. Des décalés de 0 ou 5 bits ne sont pas autorisés lorsque le PC est utilisé pour l'indexation. L'instruction :

```
LDY OFFSET,X
```

nécessite déjà un autre octet supplémentaire parce que l'opc pour LDY a 2 octets de longueur.

La plupart des assembleurs utilisent la notation spéciale PCR (compteur de programme relatif), qui fait que l'assembleur se sert du PC comme d'un registre d'index et calcule le décalé. Par exemple :

```
DATITM FCB 0
       .....
       LDA DATITM,PCR
```

Émulation de terminal

Nous donnons un sous-programme utilisant cette technique pour permettre l'émulation de divers terminaux, de sorte qu'un programme écrit pour utiliser un type particulier de terminal peut tourner sur votre système. Les différences entre terminaux sont plus apparentes dans les codes employés pour contrôler les diverses fonctions d'écran, telles que l'effacement et le positionnement du curseur. Celles-ci peuvent être des codes de contrôle (caractères dont le code ASCII est inférieur à 32) ou d'échappement, consistant en un caractère ESCAPE (ASCII 27) suivi de n'importe quel autre caractère ou séquence de caractères. Notre simple routine ne permet que la substitution d'un caractère de contrôle par un autre, ou un seul caractère suivant ESCAPE par un seul autre caractère. Mais la routine montre clairement comment mener une telle émulation. On tient deux tables : l'une contient les caractères de contrôle; l'autre les caractères d'échappement. Si un programme sort un caractère de contrôle, par exemple, alors il sert de décalé dans la table pour trouver le caractère qui devra effectivement être affiché.

Entièrement translatable, la routine peut s'adjoindre à tout autre programme en n'importe quelle position. Nous supposons l'existence d'une routine de système d'exploitation (OUTCH), qui envoie le caractère dans l'accumulateur A vers l'écran, et nous utilisons JMP pour accéder à cette routine — qui doit se trouver dans une position fixe de la mémoire. Notez que la directive ORG doit toujours être donnée, bien qu'elle n'ait pas d'effet. Le caractère à afficher doit se trouver en A.

Longueurs d'instructions

Le problème du calcul de la longueur des instructions n'est pas restreint à l'usage de l'adressage « compteur de programme relatif ». Il est souvent nécessaire de connaître la longueur totale d'une routine à insérer dans un espace mémoire restreint — par exemple dans une ROM. Tout ouvrage relatif au langage d'assemblage 6809, ou au manuel d'assembleur, doit inclure une table des mnémoniques associés aux données. Pour chaque mnémonique, cette donnée inclura un opc correspondant, la longueur totale de l'instruction (bien que cela puisse ne pas être possible, auquel cas la longueur minimale sera donnée, suivie d'un signe +), le nombre de cycles d'horloge nécessité par l'instruction, et l'effet de l'instruction sur les drapeaux de code condition.

Les règles générales pour calculer les longueurs d'instructions — et donc pour écrire un code compact — sont les suivantes :

1. La plupart des opc sont à un octet; ils affectent directement le contenu de S et Y (sauf LEA) et certains, qui affectent U (comme LDY et STS), ont 2 octets de longueur.
2. Toute adresse indexée nécessitera l'utilisation d'un après-octet, et éventuellement de 1 ou de 2 octets supplémentaires. Cela dépendra de la taille du décalé.
3. Les données suivant l'opc en mode immédiat ont 1 ou 2 octets de longueur, selon la taille du registre utilisé.
4. Les adresses doivent avoir 1 octet de longueur en mode page directe (généralement des emplacements compris entre 0 et $FF), et sinon 2 octets. Tous les assembleurs n'utilisent pas convenablement l'adressage direct; ainsi, une adresse peut être transformée en 2 octets alors qu'on n'en attendait qu'un.

Le problème du calcul du temps pris par chaque instruction est également compliqué, ne serait-ce que parce que ce temps dépend du nombre d'octets qui doivent être cherchés, c'est-à-dire de la longueur de l'instruction.

C'est important dans les applications en temps réel et pour la conduite de certains périphériques. Le temps pour chaque instruction est donné par le nombre de cycles d'horloge — ou, du moins, le nombre minimal de cycles d'horloge — que prend l'instruction; ainsi, le temps pris dépen-

dra aussi de la fréquence d'horloge. Pour un système 6809, on a couramment 1 MHz — un million de cycles par seconde. Chaque cycle d'horloge prend donc un millionième de seconde. La simple instruction :

```
LDA DATITM
```

qui utilise une adresse 16 bits prend cinq cycles, et s'exécute donc en 5 millionièmes de seconde. L'instruction : PSHS PC,B,CC prend cinq cycles plus un pour chaque octet entré sur la pile ; soit dans ce cas, un total de neuf cycles (souvenez-vous que le PC a 2 octets).

Si un système n'inclut pas une horloge temps réel, alors le seul moyen de mesurer le temps écoulé est une routine de retard logiciel. Celle-ci exécute une séquence d'instructions dont les temps individuels doivent être choisis de telle sorte que la somme donne l'intervalle requis. De tels intervalles sont habituellement mesurés en millisecondes (millièmes de seconde), et il n'est donc pas nécessaire d'être trop exact — une différence d'un millionième de seconde n'importera pas. En supposant une fréquence d'horloge de 1 MHz, la routine de retard logiciel que nous donnons ici produira des retards compris entre 1 et 255 millisecondes : le nombre exact de millisecondes (ms) étant passé comme paramètre dans A. La notation (A) signifie le contenu de l'accumulateur A.

Le calcul pour trouver la constante COUNT peut s'exprimer comme suit :

INSTRUCTION	NOMBRE DE CYCLES D'HORLOGE	NOMBRE DE FOIS EXÉCUTÉES	TEMPS PRIS (CYCLES D'HORLOGE)
PSHS B,CC	7	1	7
LDB # COUNT	2	(A)	(A) * 2
DECB	2	(A) * COUNT	(A) * COUNT * 2
BNE LOOP2	3	(A) * COUNT	(A) * COUNT * 3
DECA	2	(A)	(A) * 2
BNE LOOP1	3	(A)	(A) * 3
PULS PC,B,CC	9	1	9

Cela donne un total de (A)*(7+5*COUNT)+16 cycles d'horloge. Pour faciliter le calcul, nous ignorerons 16. A une fréquence d'horloge de 1 MHz, il y a 1 000 cycles d'horloge dans une milliseconde, donc le temps total sera : (A)*1 000 cycles d'horloge.

(A)*(7+5*COUNT) = (A)*1000
(7+5*COUNT) = 1000
5*COUNT = 973
COUNT = 195 (arrondi à l'entier le plus proche).

Il est tout à fait possible de prendre des retards plus précis et d'utiliser les registres 16 bits pour un plus grand intervalle, mais le principe de décrémentation d'un registre, un nombre fixe de fois, reste le même.

Routine d'émulation de terminal

```
ESCAPE  EQU   27
SPACE   EQU   32          (L'espace est ASCII 32.)
OUTCH   EQU               Entrer ici l'adresse du
                          système d'exploitation.
        ORG   $1000
CTABLE  RMB   32          Caractères de table de
                          contrôle.
ETABLE  RMB   128         Caractères de table
                          d'échappement.
EFLAG   FCB   0           Drapeau indiquant si le
                          dernier caractère était
                          ESCAPE.
DISPCH  PSHS  X           Sauvegarde X.
        TST   EFLAG, PCR  Vérifie si le dernier
                          caractère était ESCAPE.
        BEQ   DISP1       Si pas ESCAPE, alors va en
                          DISP.
        LEAX  ETABLE, PCR Sinon prend adresse de
                          ETABLE dans X.
        LDA   A,X         Prend caractère de remplacement
                          utilisant le caractère
                          original en A comme décalé.
        CLR   EFLAG, PCR  Réinitialise EFLAG.
        BRA   FINISH
DISP1   CMPA  SPACE       Vérifie si caractère de
                          contrôle.
        BGE   FINISH      Si pas caractère de
                          contrôle, alors va en FINISH.
        CMPA  ESCAPE      Sinon vérifie si ESCAPE.
        BEQ   ESCCH       Si c'est ESCAPE, alors va en
                          ESCCH.
        LEAX  CTABLE,PCR  Prend l'adresse de CTABLE en X.
        LDA   A,X         Prend caractère de
                          remplacement en utilisant
                          le caractère initial en A
                          comme décalé.
        BRA   FINISH
ESCCH   INC   EFLAG,PCR   Met EFLAG pour indiquer que
                          c'était un caractère
                          d'échappement.
FINISH  PULS  X           Restaure X.
        JMP   OUTCH       Affiche caractère en A.
        END               Notez que le RTS à la fin de
                          OUTCH redonne le contrôle
                          au programme appelant.
```

Routine de retard logiciel

```
                          Sous-programme pour
                          retarder de (A)
                          millisecondes.
COUNT   EQU   195         Voit calcul.
        ORG   $1000
DELAY   PSHS  B,CC        Sauvegarde les deux autres
                          registres affectés.
LOOP1   LDB   #COUNT      Compte 1 ms.
LOOP2   DECB              Continue à décrémenter.
        BNE   LOOP2       Jusqu'à ce que B devienne
                          nul.
        DECA              Décrémente A après
                          chaque ms.
        BNE   LOOP1       Jusqu'à ce que A devienne
                          nul.
        PULS  PC,B,CC     Retour.
```

Attaque psychique

Les jeux d'arcades de type « combats galactiques » sont en perte de vitesse. Aujourd'hui, les logiciels à succès mêlent des éléments très divers : arcades, stratégie, et même jeu d'aventures.

Psytron est un jeu aussi complexe que passionnant; le graphisme en est remarquable, et vous aurez besoin de bien du temps avant de vous en rendre maître.

Vous jouez le rôle de Psytron, organisme mi-humain, mi-informatisé, qui contrôle une colonie spatiale sur la planète Betula 5.

Elle comprend divers bâtiments, dont chacun a une fonction spécifique : c'est ainsi que les systèmes de culture hydrologiques sont vitaux pour les colons, tandis que les installations productrices d'énergie permettent à l'ordinateur de fonctionner. La colonie serait sans défense si la principale d'entre elles était mise hors d'usage. L'emploi des touches du clavier, ou d'un manche à balai, donne au joueur une vue à 360° de toutes les différentes constructions; le graphisme est très soigné et donne à l'ensemble une allure vraiment réaliste.

A chaque niveau du jeu, Psytron ressemble de très près à ce que proposent habituellement les jeux d'arcades. Le joueur dispose de plusieurs armes de façon à parer aux coups des envahisseurs étrangers. Ce n'est qu'au quatrième niveau que les éléments stratégiques du jeu commencent à devenir véritablement apparents.

Au premier niveau de jeu, Psytron contrôle un « droïde », dont le rôle est de détruire des saboteurs tripodes envoyés par l'ennemi pour endommager les installations de la base.

Au deuxième et au troisième niveau, vous devrez faire face aux soucoupes volantes de l'adversaire : vous pourrez les abattre une par une, ou faire usage du « disrupteur », qui annihile tous les appareils en vue; mais il est instable et a environ une chance sur dix d'exploser quand il est mis en service...

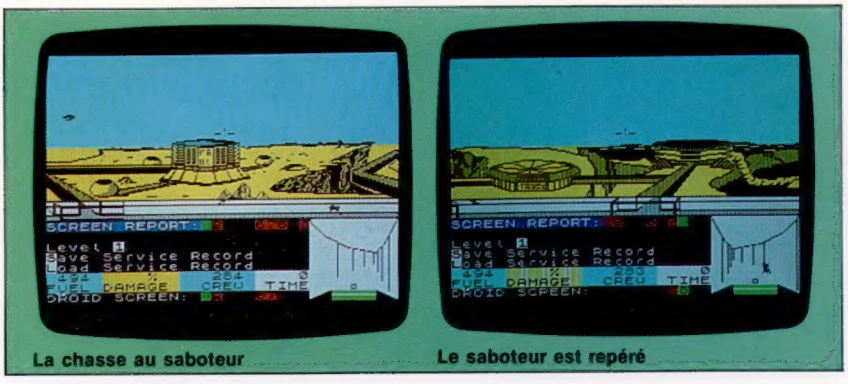
Parvenu au quatrième niveau, vous serez amené à faire certains choix stratégiques, en fonction des destructions imposées à vos installations. Les extraterrestres ne cessent d'attaquer les points vitaux de votre base et de lancer des saboteurs dans des missions-suicides; mais c'est ici qu'intervient l'option *Freezetime,* qui vous permet de « geler » l'action par simple appui de la touche RETURN. Vous recevez alors des rapports qui vous renseignent sur les dommages qui vous ont été infligés. Vous aurez à prendre en compte le nombre d'hommes d'équipage tués ou blessés, les approvisionnements qui vous restent, et l'état de l'usine productrice d'énergie. Vous pourrez procéder à des réparations et affecter vos hommes aux endroits où ils seront le plus efficace. Cela exige une répartition minutieuse des ressources — une équipe de réparation consomme plus d'oxygène et de nourriture, et il sera parfois nécessaire d'abandonner certains bâtiments dans le but de conserver assez de carburant, d'air et de ravitaillement.

Au cinquième niveau, il devient possible de communiquer avec un vaisseau de secours, qui pourra vous faire parvenir ce dont vous avez désespérément besoin — à condition que la piste d'atterrissage soit encore en état de fonctionnement...

Au sixième et dernier niveau, c'est l'aspect stratégique qui prend définitivement le dessus. A ce stade, le seul objectif est de tenir encore pendant une heure, et de conserver la maîtrise de la base en tirant parti des ressources que vous avez amassées lors des niveaux précédents. L'action s'accélère encore, les assaillants sont de plus en plus nombreux, et vous vous rendrez vite compte qu'il est impossible de tenir toutes les installations face à la supériorité écrasante de vos adversaires. Vous serez donc contraint de sacrifier certains bâtiments — mais surtout pas la piste d'atterrissage, puisque c'est seulement par là que les secours peuvent vous parvenir.

Il s'est décidément passé bien du temps depuis l'apparition de Space Invaders. Et Psytron, aussi difficile que passionnant, marque un gros progrès par rapport aux jeux d'arcades classiques qui ont si longtemps dominé le marché.

L'œil du droïde
Ces deux images extraites de Psytron correspondent au premier niveau du jeu. A mesure que le droïde pourchasse le saboteur à travers les corridors, la vue d'ensemble se modifie; une fenêtre (en bas et à droite de l'écran) permet au joueur de « voir » à travers les yeux du droïde. Lorsque le saboteur est enfin repéré (image de droite), il peut être détruit par simple pression sur le bouton de mise à feu.
(Cl. Spectrum.)



La chasse au saboteur — Le saboteur est repéré

Psytron : pour Spectrum 48 K et Commodore 64.

Éditeurs : Beyond Software.

Auteurs : Tayo Owolu et Paul Voysey.

Manche à balai : optionnel.

Format : cassette (Spectrum), cassette ou disquette (Commodore 64).

Page manquante
(publicité)

Page manquante
(publicité)